Titre: Title:	Détection des points d'intersection de rues dans les cartes géographiques
Auteur: Author:	Giang Do-Tien
Date:	1997
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Do-Tien, G. (1997). Détection des points d'intersection de rues dans les cartes géographiques [Mémoire de maîtrise, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/6714/

UNIVERSITÉ DE MONTRÉAL

DÉTECTION DES POINTS D'INTERSECTION DE RUES DANS LES CARTES GÉOGRAPHIQUES

par

Giang DO-TIEN

DÉPARTEMENT DE GÉNIE ÉLECTRIQUE ET INFORMATIQUE

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION

DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES

(GÉNIE ÉLECTRIQUE)

MAI 1997

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé

DÉTECTION DES POINTS D'INTERSECTION DE RUES

DANS LES CARTES GÉOGRAPHIQUES

présenté par : Giang DO-TIEN

en vue de l'obtention du diplôme de : Maîtrise ès sciences appliquées (M. Sc. A)

a été dûment accepté par le jury d'examen constitué de :

HERVÉ, Jean-yves, Ph.D., président

PLAMONDON, Réjean, Ph.D., membre et directeur de recherche

LI, Ying, Ph.D., membre externe du Centre de Recherche Informatique à Montréal

À ma tante, Lan Phuong Ly, son mari, Tuong Trieu Nguyen,

et à tous les membres de sa famille qui m'ont permis d'atteindre ce niveau.

À mes parents Hong Phan Ly et Duc Ngoc Do et à mes soeurs

qui m'ont supporté durant ces derniers jours.

À mon amie de cœur Pham Ngoc Anh Ha,

pour ses supports sans fin et ses beaux sourires éternels.

REMERCIEMENTS

Je tiens tout d'abord toute ma gratitude envers mon directeur de maîtrise, le Dr. Réjean Plamondon, pour ses encouragements et son esprit ouvert pendant la réalisation de ce projet. Grâce à son sens optimiste et ses vastes connaissances dans le domaine de la reconnaissance de forme, il m'a donné des guides très précieux notamment au niveau des documentations bibliographiques et au niveau des méthodes de travail.

Je tiens également à remercier les membres industriels tels que la compagnie M3i, le Centre de Recherche Informatique à Montréal, et l'ATS Aérospatiale pour leur initiative à la création des projets de recherche intéressants et pour leur aide financière importante.

Je voudrais remercier également tous les membres, les amis et les amies du laboratoire Scribens qui ont partagé avec moi des moments inoubliables.

Je tiens à remercier tous les membres du jury et particulièrement M. Patrice Hébert de la compagnie M3i, pour avoir apporté leur appréciation à ce travail.

Finalement, et par dessus tout, je voudrais remercier infiniment mon amie de cœur, Pham Ngoc Anh Ha, qui m'a aidé à traverser les moments difficiles de ces dernières années, particulièrement lors de la rédaction de ce mémoire.

RÉSUMÉ

Notre projet de recherche porte sur la mise au point d'algorithmes qui détectent les intersections de rues dans une image de carte géographique et il fait partie d'un projet industriel d'envergure traitant et analysant les cartes. Les images des cartes que l'on traite sont des images en noir et blanc de format TIFF. Les rues qui apparaissent dans l'image devraient avoir une largeur visible de l'ordre de 10 à 15 pixels pour être détectées correctement. Nous évitons donc de détecter les intersections des petites rues.

Puisque chaque rue est composée de deux segments droits séparés entre eux par une distance de 10 à 15 pixels, une intersection de rues sera formée nécessairement par les croisements de quatre segments (deux rues). Alors, l'intersection recherchée devrait se situer approximativement au centre de masse formé par les quatre points de croisements.

Pour trouver ces points, nous devons être capable s'identifier (paramétriser) les quatre segments qui forment les deux rues. Pour trouver ces segments, il faut connaître a priori le lieu approximatif du point d'intersection de ces segments dans l'image considérée. Étant donné qu'une même intersection de rues peut exister à différents endroits sur différentes cartes, il faut donc retrouver ces intersections de rues à l'aide des références géographiques qui sont invariantes indépendamment des cartes. Ainsi, la rue AAA devrait intercepter la rue BBB à des coordonnées géographiques fixes malgré leur position graphique variable d'une carte à une autre.

Étant donné que les informations géographiques sont essentielles pour la détection des coins d'intersection de rues, nous voulons que ces informations soient disponibles dans un fichier de référence que nous nommons BDG (base de données globale). Pour fin de simulation, nous avons préparé ce fichier sous format de texte de façon à ce que chaque ligne possède les informations suivantes :

Nom de la rue 1	*Nom de la rue 2*	*Longitude °*	*Latitude °*

Pour évaluer les coordonnées graphiques de la région dans laquelle nous voulons trouver une intersection de rues, il nous faut des formules de conversion des coordonnées géographiques en coordonnées graphiques et vice versa :

$(u,v) \leftrightarrow (i,j)$

où u est la longitude, v est la latitude, i est la coordonnée graphique en termes de colonne et j est la coordonnée graphique en termes de rangée. Ces formules de conversion devront être déduites à partir des paires de points initiaux $[(u_n, v_n),(i_n, j_n)]$ (où n est un petit entier) servant comme cas de références utilisant la méthode de triangulation. Ces formules devront satisfaire entièrement aux conditions initiales de ces cas de références et être capables d'interpoler la réponse si l'on fournit des données qui ne se trouvent pas parmi ces références.

Les références que nous choisissons pour la génération des formules de conversions sont les coordonnées géographiques des quatre coins extrêmes de la carte ($C_{nu} \equiv u_n, C_{nv} \equiv v_n$) et les coordonnées graphiques de ces coins ($C_{ni} \equiv i_n, C_{nj} \equiv j_n$). On voudrait faire en sorte que l'ordinateur soit capable de nous fournir automatiquement les coordonnées graphiques des quatre coins extrêmes de la carte, comme tout humain peut le faire en regardant l'image de carte et en pointant le coin avec un souris. La seule chose que nous ne pouvons pas automatiquement déterminer ce sont les coordonnées géographiques correspondant aux quatre coins de la carte. Ces dernières doivent être fournies a priori à partir d'un fichier que nous considérons comme base de données relatives (BDR). Dans ce fichier, se trouvent toutes les informations pertinentes, mais relatives à une image de carte, notamment l'information sur l'échelle de l'image [km/pouce], la résolution de l'image [pixels par pouce], les coordonnées géographiques

des quatre coins extrêmes de la carte, etc. Ces données sont suffisantes pour effectuer un alignement global de l'image de carte en supposant que les images soient rectangulaires.

Nous voyons apparaître deux options pour la détection des coins. La première option (Option≡MAP_INSIDE) concerne la détection des coins résultant des intersections de rues à l'intérieur de la carte et la deuxième concerne la détection des coins extrêmes des cartes.

Chaque type de détection requiert des considérations différentes, car chaque type de coins possède, de par sa nature, des caractéristiques différentes. Par exemple, les coins extrêmes sont le résultat de l'intersection de deux segments seulement : l'un est horizontal et l'autre est vertical, tandis que les points d'intersections de rues, selon notre convention, devraient être déduits à partir des croisements de quatre segments qui sont parallèles deux à deux entre eux. L'étude de l'ensemble de ces caractéristiques nous aide énormément dans la définition des conditions initiales (contraintes) afin d'optimiser les algorithmes.

Parmi les algorithmes développés, l'étape la plus importante est la détection des lignes principales. Pour cela, nous utilisons la transformée de Hough modifiée selon une nouvelle méthode d'implantation. La transformée de Hough est reconnue pour être imprécise et demande beaucoup de temps de calcul et d'espace de mémoire. Nous avons démontré dans ce travail que la transformée de Hough, implantée selon notre nouvelle méthode, demande moins d'espace de mémoire tout en donnant des résultats précis.

Ainsi, pour que la transformée de Hough soit efficace, il faut que l'image soit la plus propre possible, c'est-à-dire qu'elle ne doit contenir que des segments de lignes de la largeur d'un pixel. Il faut donc un bon algorithme de détection de contours et d'amincissement. Pour atteindre ce but, nous présentons un nouvel algorithme de détection de contours qui a été récemment développé par des chercheurs australiens.

C'est un algorithme qui modélise le système de détection de contours chez l'être humain. Nous démontrons par ailleurs qu'en choisissant des paramètres appropriés, ce même détecteur peut aussi faire de l'amincissement pour les lignes de largeurs de quatre pixels ou moins.

Tous les algorithmes discutés ci-dessus sont testés avec 27 images réelles de format TIFF de grandeur 1000 x 1000 pixels pour la détection des coins extrêmes et avec deux images réelles pour la détection des points de l'intersection de rues.

Tel qu'implanté, l'accumulateur de Hough occupe une taille moyenne de 100 casiers (éléments), chacun avec des paramètres (d,θ) très près des quantités estimables par règle. La quantité d'erreur en terme de pixel pour la détection de quatre coins extrêmes de carte est de 1.2 pixels et pour la détection des intersections de rues est de 7 pixels. Les modules sont écrits en langage C++, compilés en gcc dans Sun-UNIX pour être entièrement compatibles avec le système global, un nom de IIT (interactive image technologies), développé par notre partenaire : le Centre de Recherche Informatique de Montréal (CRIM) en collaboration avec M3i et ATS Aérospatial.

ABSTRACT

Our project deals with the development of algorithms that detect intersections in a road map and is part of a large industrial project dealing with map analysis and recognition. Those maps must be in black and white saved in the TIFF format. Roads which appear inside the image must have a width about 10 to 15 pixels to be detected easily. Detecting intersections from small roads is avoided.

As each road is represented by two linear segments which are separated by a distance of 10 to 15 pixels, the intersection of two roads must be determined by crosses from four linear segments (or 2 roads). Once the position of these four points is determined, the intersection is assumed to be located approximately at the center.

To determine those points, we must be able to identify the four segments that form the two roads. To find those segments, we must have a general idea of the position in the image where those segments could be found. Since the same intersection could exist at different places on different maps, we must use geographical references of roads' intersection to get into the approximate place in the map where we can find out that roads' intersection. Such references can be considered as invariant, with respect to the map that we are considering. For example, the road named AAA must intersect the road named BBB at a fix geographical coordinates even if its graphical position changes from one map to another.

The geographical information is so important that it requires a complete file in the whole system to store only geographical coordinates of all roads' intersections. For the simulation associated with this specific project, we prepare a file in text format such that each line has the convention as follows:

Road's name 1	*Road's name 2*	*Longitude °*	*Latitude °*

Once we have selected a geographical coordinate of the intersection of two given roads, we must have a way to convert this information to its equivalent graphical position inside an image of a map. For that, we must develop formulas to convert geographical coordinates to graphical coordinates and vice versa.

$(u,v) \leftrightarrow (i,j)$

where u is the longitude, v is the latitude, i is the graphical coordinate in column and j is the graphical coordinate in row. These formulas must be determined from pairs of initial points $[(u_n,v_n),(i_n,j_n)]$ (where n is a small integer) using triangulation method. Those pairs are considered as reference cases. These formulas should satisfy entirely to initial conditions imposed by reference cases and must be able to interpolate all other cases that do not appear in the reference cases.

We have chosen the geographical coordinates of the four extreme corners ($C_{nu} \equiv u_n, C_{nv} \equiv v_n$) of the map and the graphical coordinates of four extreme corners ($C_{ni} \equiv i_n, C_{nj} \equiv j_n$) as reference points. Our objective is to build an algorithm that can detect automatically graphical positions of the four extreme corners of a map, as a human would do when he looks at the image and points to those corners with a mouse to notify their graphical position. The only thing that we cannot determine automatically is the geographical coordinates of the four extreme corners of the map. This information must be stored in another text file that we consider as a relative data base. This file includes all the pertinent information that are relative to an image of a map, say the image scale [km/inch], the image's density [dot/inch], the geographical coordinates of the quatre extreme corners of the map, etc. This information is enough for warping the map globally if the map was rectangular.

Two kinds of corners must be detected by our algorithm. A first option (Option≡MAP_INSIDE) allows the detection of corners formed by road's intersection

and a second (OPTION≠MAP_INSIDE) deals with the detection of extreme corners of a map.

Each kind of corners requires a distinct method of detection because they are of different nature. For example, an extreme corner is the result of a intersection between only two segments : one horizontal and one vertical, but the position of a road's intersection, with roads having the properties as defined before, must be determined by intersections from four segments that are parallel two by two. A deep study in those properties helps us a lot in determining the initial conditions to optimize the design of our algorithmic.

Among the algorithms that we have built, the most important is the algorithm for detecting principal lines in an image. For that, we use a new version of Hough transform that we specially designed to improve the performance of line detection because the usual method of implantation of Hough transform is known to be inaccurate and requires a lot of memory space. We have shown, in this work, that the Hough transform, as implemented by our new method, uses less memory without losing precision.

However, to assure the performance of Hough transform method, we must also make sure that the image to analyse is « clean ». It means that the image must contain only lines of 1 pixels thickness. For this purpose, first of all, we filter the image with an edge detector proposed by Morrone and Burr [1]. This operator closely mimics the property of human eyes because it is based on the fact that human eyes use some kinds of filter to detect edge. We described in this conference how Morrone and Burr proved that human eyes use principally two kinds of filter : one with even symmetric field and the other with odd symmetric field. We also illustrated that by choosing proper values for the parameters of the operator, this edge operator can also act as a thinning algorithm for lines that have a thickness smaller than quatre pixels.

The algorithms that we developed were tested on 27 real images in TIFF format of size of about 1000x1000 pixels for the detection of the quatre extreme corners. Two real images were also used for the test of road's intersection detection.

As implemented, the average effective size of the Hough accumulator in used is 100 with parameters having very small difference compared to what we can approximate by ruler. The amount of errors in term of pixel for the detection of the quatre corners of the map is in average 1.2 pixel and for the detection of intersections inside the map is 7 pixels. These modules are written in C++ language, they are compiled with gcc in UNIX so that they can be integrated entirely in a global system, named IIT (interactive image technologies), developed by « le Centre de Recherche Informatique de Montréal » (CRIM) with the collaboration of M3i and ATS Aerospace.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

Chapitre 1

Introduction

1.1 MISE EN CONTEXTE

À l'aube de l'an 2000, alors que presque tous les processus industriels sont maintenant informatisés, plusieurs villes souhaitent aussi automatiser leurs opérations en ce qui concerne la gestion des zones urbaines. Pour cela, on voudrait emmagasiner tous les détails géographiques d'une ville dans un ordinateur. Idéalement, on exige que l'ordinateur soit capable de fournir non seulement l'image de la carte d'une région, mais aussi n'importe quel détail civil.

On parle donc de la reconnaissance automatique de dessins sur une carte ou sur une image de carte. Pour obtenir l'image d'une grande carte, il faut numériser plusieurs cartes et les assembler. La numérisation des cartes entraîne une distorsion locale dans l'image, tandis que l'assemblage des images de carte cause de la distorsion globale sur l'image finale.

Il est important que ces images contiennent le moins de distorsions possible pour que les opérateurs puissent y ajouter facilement d'autres détails civils qui apparaîtront dans le futur, par exemple lorsqu'un opérateur reçoit la recommandation d'ajouter un symbole représentant un nouvel immeuble à des coordonnées géographiques (A° latitude, B° longitude) ; si la carte contenait encore des distorsions, l'opérateur risquerait de placer le nouvel immeuble à un mauvais endroit sur l'image. Ce genre d'erreur est encore plus facile à commettre lorsque nous attribuons cette tâche à une machine.

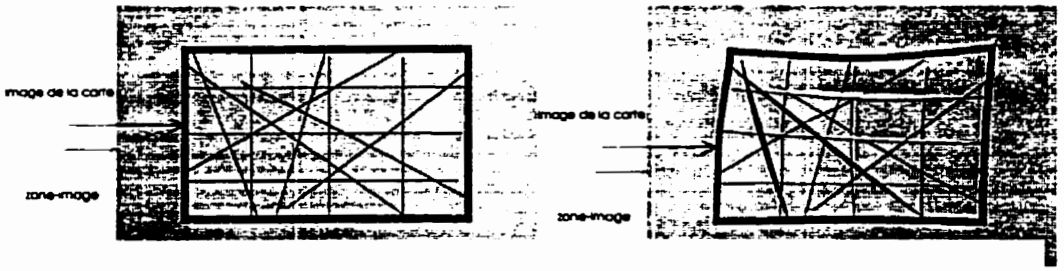
Évidemment, l'être humain ne sera jamais capable d'évaluer le degré de distorsion d'une image et de faire les corrections nécessaires, car une image de carte en géographie a souvent une grandeur énorme. De plus, il faut parfois faire ces tâches sur de nombreuses cartes pour couvrir une grande région. On voit donc la nécessité d'utiliser des ordinateurs pour opérer ces tâches longues et lourdes.

Dans ce contexte, M3i et ATS Aérospatiale se sont associés avec le CRIM et avec plusieurs chercheurs universitaires pour développer un système complet de cartographie numérique. Le présent projet de maîtrise s'inscrit dans ce contexte et porte sur les étapes initiales de corrections automatiques des distorsions.

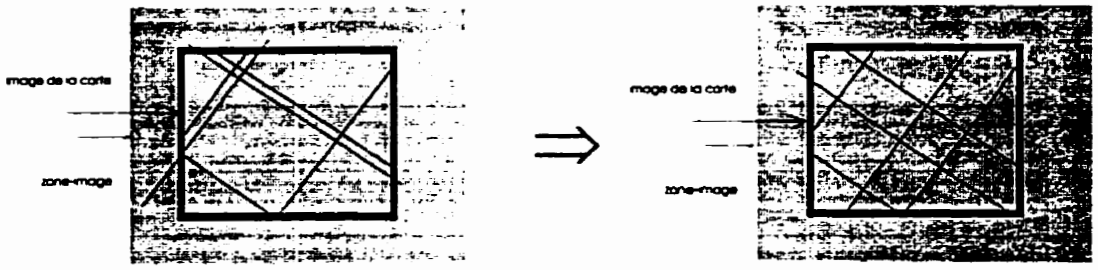
Plus précisément, notre projet consiste à trouver et à implanter des algorithmes pour corriger les distorsions dans les images de cartes géographiques. La numérisation d'une carte quelconque donnera certainement une image plus ou moins bruitée (distordue). Même si ces distorsions, que l'on appelle distorsions naturelles, sont de l'ordre de quelques pixels seulement, elles deviennent considérables lorsque qu'on arrive à l'étape d'assemblage de ces images pour obtenir une plus grande carte.

De plus, dans les anciens travaux de construction des cartes géographiques, les concepteurs de cartes ont parfois intentionnellement écarté, rétréci ou déplacé davantage une information quelconque dans une carte pour pouvoir y insérer des notes ou des

légendes. Ces transformations ont donné lieu à un nouveau type de distorsions, que l'on appelle pour l'instant la distorsion forcée, et qui constitue souvent une source d'erreur beaucoup plus grave par rapport à la distorsion naturelle (figure 1.1).



(a) Image fidèle à la réalité versus image avec distorsions naturelles



(b) Image fidèle à la réalité versus image avec distorsions forcées

Figure 1.1 Illustration des types de distorsions

Pour réduire ces distorsions et redresser l'image, il nous faut développer une ou plusieurs formules qui transforment un point graphique *(i,j)* en un autre point *(i',j')*. La raison d'être de ces transformations n'est rien d'autre que d'appliquer un facteur de correction sur chacun des points graphiques *(i,j)* de l'image source pour que chacun des points *(i',j')* de l'image destinataire soit déplacé dans les proportions qui varient selon les degrés de distorsion locale.

Pour l'instant, on peut modéliser ces transformations par les relations suivantes :

$$i' = F(i, j) \quad (1.1)$$

$$j' = G(i, j) \quad (1.2)$$

Afin de trouver la fonction F et G, il nous faut des conditions initiales. Ces conditions (contraintes) sont définies à partir de deux ensembles de points représentatifs. Le premier ensemble contient des points graphiques *(i,j)* de l'image source qui devraient être déplacés aux points graphiques *(i',j')* de l'image destinataire. Le deuxième ensemble contient les points *(i',j')* qui correspondent un à un à chaque point *(i,j)* du premier ensemble.

Disposant de ces deux ensembles de points, nous pouvons trouver plusieurs solutions pour *F* et *G*. Certaines solutions nécessitent plusieurs étapes de calcul pour arriver à des fonctions *F* et *G* raisonnables. D'autres s'appliquent mal à certains points *(i,j)* de l'image, par exemple dans le cas où le point *(i',j')* correspondant se situera à l'extérieur de l'image. D'après l'étude de Detlef et Heinrich (1995), la méthode de triangulation est la meilleure méthode pour trouver des fonctions *F* et *G* optimales. Une fois que *F* et *G* sont déterminées, le travail d'alignement est presque terminé, car il ne reste plus qu'à appliquer ces transformations à chacun des points graphiques de l'image source. Cette étape est représentée par le module 3.0 nommé « Aligner » dans le premier diagramme de flux de données illustré à la figure 2.2. Cette étape a été complétée par l'équipe du CRIM dès les premiers jours du projet (CRIM, 1995).

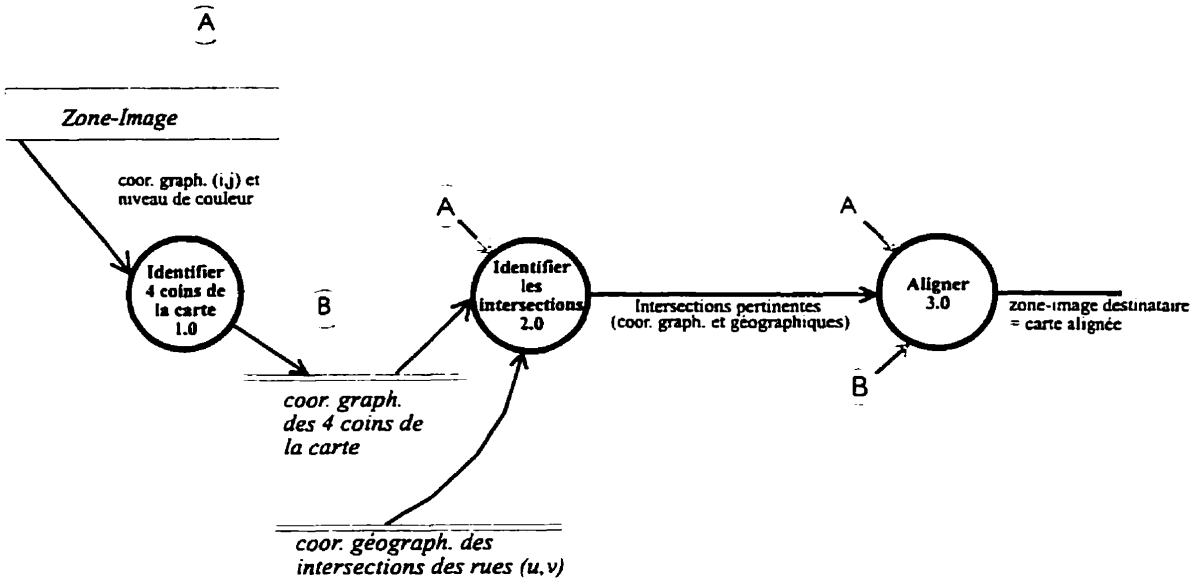


Figure 1.2 ***Diagramme de flux global du mécanisme d'alignement automatique***

Le défi à relever pour réussir à appliquer cette méthode réside au niveau de la construction automatique des deux ensembles de points *(i,j)* et *(i',j')*. Pour arriver à dire que le point graphique situé à la position (31,400) devrait être au point (39,405) par exemple, il faut une bonne référence. Cette référence peut être sous forme d'image (une autre vue de la carte), sous forme de données graphiques ou sous forme de données géographiques.

Parmi ces types de référence, la référence la plus fiable est sans doute la référence géographique des objets représentés sur la carte, car elle est toujours la même indépendamment des cartes à analyser.

Ainsi, l'intersection des rues Jean-Talon et Papineau doit être nécessairement située à des coordonnées géographiques précises (u^O, v^O) et ce, indépendamment de la carte que l'on est en train de traiter.

Dans le projet global, ces références géographiques seront emmagasinées dans une base de données, nommée BDG, qui contient entre autres les informations suivantes :

Nom de rue 1	*Nom de rue 2*	*Longitude(°)*	*Latitude (°)*
Jean-Talon	*Papineau*	*xxx.xxxxxxxx*	*yyy.yyyyyyyy*

Évidemment, il faut aussi des règles de transformation pour pouvoir utiliser efficacement ces données de référence, car les deux ensembles à construire sont exprimés en coordonnées graphiques et non géographiques. Heureusement, il y a un lien étroit entre les données référentielles et les points graphiques d'une image de carte. En effet, l'intersection des rues Jean-Talon et Papineau, par exemple, peut être vue en haut à gauche d'une carte, tout comme elle peut être vue au centre d'une autre carte. Il n'y a donc pas de correspondance fixe entre les données référentielles et les points graphiques d'une carte. Ces correspondances sont propres à chaque carte.

Il faut alors quelques références supplémentaires décrivant les liens entre un certain nombre de coordonnées géographiques et un certain nombre de points graphiques de la carte à traiter. Autrement dit, il faut aussi trouver quatre formules décrivant les liens suivants :

$$i = A(u,v) \tag{1.3}$$
$$j = B(u,v) \tag{1.4}$$

et

$$u = a(i,j) \tag{1.5}$$
$$v = b(i,j) \tag{1.6}$$

Encore une fois, si l'on veut utiliser la méthode de triangulation pour déterminer les fonctions A, B, a, b, il est nécessaire et suffisant d'utiliser quatre points de référence. Ainsi, toute image que notre système aura à traiter devrait avoir :

1. une référence contenant des coordonnées géographiques des intersections de rues qui se retrouvent dans l'image de carte et

2. quatre coordonnées géographiques des quatre coins extrêmes de la carte.

Nous avons choisi les quatre coins extrêmes de la carte comme références, car théoriquement, ces coins sont uniques pour une carte rectangulaire donnée. Connaissant ces références, on peut déterminer les fonctions A, B, a, b avant de commencer à construire les deux ensembles de points discutés ci-dessus. Cette étape peut être nommée l'étape de déduction des formules de conversion des coordonnées géographiques à des coordonnées graphiques et vice versa.

Une fois les formules de conversion déterminées, nous pouvons prendre une référence géographique quelconque et la convertir en un point graphique correspondant. Si l'image de la carte n'est pas distordue à cette position, le point que l'on a trouvé devrait correspondre visuellement à une intersection de rue. Dans le cas contraire, le point trouvé sera plus ou moins déplacé par rapport au point d'intersection de rue observable dans l'image. À l'œil nu, un humain peut identifier ce déplacement auquel cas il peut aisément noter la position du point destinataire. Les deux ensembles discutés auparavant peuvent donc être construits facilement par un être humain. Cependant, il en est autrement si l'on veut utiliser un ordinateur pour effectuer ces tâches. Il faut alors créer un mécanisme pour identifier la position du point destinataire à partir d'un point source. La section 1.3 présente brièvement les méthodes utilisée pour concevoir un tel mécanisme et nous allons y revenir en détail aux chapitres suivants.

Évidemment, ce mécanisme n'est pas facile à concevoir. Il est d'abord nécessaire de fixer des contraintes. Ensuite, il faut envisager des méthodes de résolution qui permettront de déterminer les inconnues.

1.2 LES CONTRAINTES SUR LES RESSOURCES DISPONIBLES

Nous venons de voir que pour que les algorithmes d'alignement fonctionnent bien, les paramètres d'entrée doivent respecter les contraintes suivantes:

Contrainte 1 - La carte à analyser doit être encadrée dans un cadre rectangulaire.

Contrainte 2 - Une base de données doit accompagner cette carte. Cette base contiendra les coordonnées géographiques des intersections de rues figurant sur la carte.

Contrainte 3 - Il faut pouvoir disposer de l'information sur l'échelle ***S_map*** de la carte à analyser (km/pouce).

Contrainte 4 - Il faut connaître l'information sur la résolution de la carte, ***PPP*** **(pixel par pouce ou Dot Per Inch)**.

Contrainte 5 - Il faut pouvoir disposer de l'information sur les coordonnées géographiques des quatre coins extrêmes de la carte selon un ordre *C0, C1, C2, C3* donné.

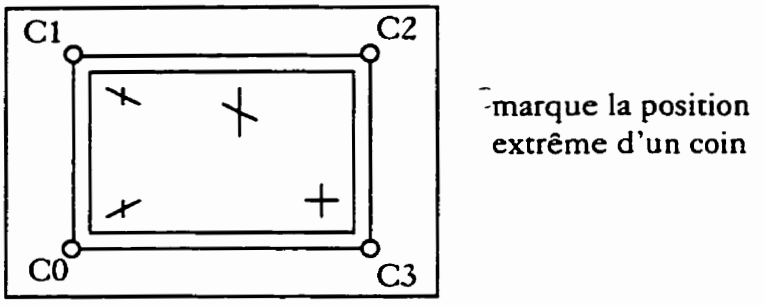


Figure 1.3 ***La disposition des quatre coins extrêmes de la carte***

Toutes ces informations doivent faire partie d'une base de données BDR, préalablement établie.

Contrainte 6 - Les rues qui entrent dans le processus de détermination des intersections de rues doivent avoir des largeurs plus grandes que sept pixels (boulevard, avenue, artères principales). Cela implique que les intersections de petites rues ne seront pas considérées. Pour assurer cette contrainte, nous supposons que le mécanisme de recherche de la base de données géographiques fait le nécessaire pour ne donner comme entrée que de bons candidats.

Contrainte 7 - La carte doit être orientée verticalement nord-sud, l'est étant de ce fait à droite.

Contrainte 8 - L'image de la carte doit être du type noir et blanc. Les pixels noirs sont appelés les pixels d'événements, tandis que les pixels blancs sont considérés comme les pixels de l'arrière-scène.

Évidemment, parmi les contraintes décrites précédemment, certaines ne sont pas encore expliquées. Elles seront introduites graduellement tout au long des chapitres.

Les contraintes étant définies, on peut maintenant décrire les méthodes envisagées. Cependant, afin de faciliter les démarches, nous avons établi une convention

sur des symboles que nous utiliserons tout au long de ce mémoire. Nous jugeons qu'il est opportun de présenter cette convention ici :

- les variables ou les indices *u, Col(lonne), x, lon(gitude), i* désignent l'axe horizontal,
- les variables ou les indices *v, Rangée, y, lat(itude), j* désignent l'axe vertical,
- les variables *Rangée, Colonne, i,j* sont des nombres entiers (coordonnées graphiques),
- les variables *u, v, x, y* sont des nombres réels (coordonnées géographiques).

1.3 MÉTHODES ENVISAGÉES

Nous avons déjà dit qu'il fallait établir des formules de conversion *A, B, a, b* et les formules d'interpolation *F* et *G* pour aligner l'image entière. Évidemment, il faut d'abord trouver les formules de conversion. Pour ce faire, il est indispensable de trouver un algorithme qui fait de la détection des coins. Un coin peut être visible ou invisible selon l'échelle où l'on regarde, alors un coin évident devrait être détectable à travers plusieurs échelles. Sachant cette propriété, plusieurs groupes de chercheurs (Chen C.-H. et al., 1995), (Fermuller C. et Kropatsch W., 1994), (Lee J.-S. et al., 1993), (Meer P., Baugher E.S et Rosenfeld A., 1988) utilisent les ondelettes (filtres de Gabor) pour essayer d'extraire des événements qui se présentent dans plusieurs échelles. Cette méthode prend beaucoup de temps de calcul et demande beaucoup d'espace de mémoire pour instaurer des liens d'analyses entre les couches de filtrage. Cette approche est idéale seulement lorsque les coins dont on cherche se manifestent à des fréquences comprises dans la largeur de bande couvert par les filtres de Gabor. Or, généralement nous ne connaissons pas a priori ces informations, il est donc plus approprié d'analyser l'image

selon l'approche spatiale. La méthode consiste à associer à tous les segments (rectilignes par morceau), qui se trouvent dans l'image, un poids représentant le nombre de pixels qui forment ces segments. Le segment ayant le poids le plus grand implique qu'il est plus facile à identifier. Ainsi, les intersections résultant de tels segments devront avoir aussi un grand poids total. On voit donc la nécessité d'employer un détecteur (quantificateur) de lignes. Pour cela, nous utilisons l'algorithme de Hough, car cet algorithme est très efficace envers des images bruitées (contrairement à l'algorithme SLIDE de Hamid K. et al., 1994). Nous verrons au chapitre 3 comment l'utilisation de la transformée de Hough (légèrement modifiée) permet de réaliser cet objectif.

Pour quantifier les lignes droites, il faut concevoir des détecteurs de ligne et de contours. Étant donné que dans une image de carte, on retrouve abondamment des lignes droites de largeur de moins de quatre pixels, les filtres de Gaussien purs (Canny, 1986), (Park, D.J. et al., 1995) ne sont pas assez efficaces. En poursuivant cet objectif, nous avons appris qu'il existait un modèle inspiré d'un modèle de perception des êtres vivants qui permet de détecter des lignes et des contours. Ce modèle, développé par l'équipe de M. Morrone (Morrone et al., 1989), sera présenté au chapitre 2.

Après avoir présenté ces deux outils indispensables, nous parlerons des méthodes d'implantation algorithmique au chapitre 4 et nous illustrerons nos résultats d'application au cinquième chapitre, en discutant des avantages et des inconvénients des méthodes envisagées. Nous ferons également référence au système complet intégrant nos algorithmes à ceux développés par l'équipe du CRIM. Enfin, nous conclurons notre travail au chapitre 6.

Chapitre 2

Modèle de Morrone

Opérateur de détecteur d'arêtes révolutionnaire

2.1 OBJECTIF

D'abord en tant qu'être humain, mais aussi en tant qu'être scientifiques, nous sommes tous émerveillés par l'architecture et la performance de notre propre système visuel. Cependant, malgré des efforts de recherche soutenus, nous ne maîtrisons pas complètement les fonctions que ce système exécute pour arriver à reconnaître des images après une courte pause d'observation. La seule hypothèse plausible et facile à vérifier est que l'être humain possède un mécanisme de détection d'arêtes. En effet, de nombreux travaux de recherche en neuropsychologie et en vision (Tolhurst 1972, Shapley et Tolhurst 1973, Hubel et Wiesel 1977, David et Morrone et al. 1989) ont permis de constater qu'il existe vraisemblablement des filtres spécialisés en matière de détection d'arêtes dans le système visuel de l'être humain et de certains autres mammifères.

D'un autre côté, les opérateurs mathématiques utilisés pour détecter les arêtes dans une image sont réputés pour être sensibles aux bruits et très dépendants de la largeur de bande de l'image. Il faut donc pratiquement définir un filtre pour chaque échelle de représentation donnée (D. Marr et al. 1980, Canny 1986). Sachant cela, nous avons décidé d'effectuer une recherche de filtres appropriés en s'inspirant des filtres qui existent vraisemblablement dans notre système visuel pour détecter les contours.

Ce chapitre est divisé en sept sections dont celle-ci en constitue la première. Dans la section 2, nous allons définir les types d'arêtes que l'on vise à détecter et la propriété idempotente des filtres de détecteur. À la section 3, nous allons présenter les synthèses des travaux de recherches de D. Marr (1982) et de Morrone et Burr (1989) (d'où vient évidemment le nom du modèle de Morrone), qui conduisent à la conclusion de l'existence des filtres de détecteur d'arêtes chez l'être humain. À la quatrième section, nous allons présenter les démarches que Christian Ronse (1993) a effectué pour trouver les propriétés mathématiques que les filtres de détecteurs, tels que supposés par Morrone, devraient posséder pour bien détecter tous les types d'arêtes dans une image. À la section 5, nous allons parler de la manière dont il faut interpréter les résultats de filtrage par les filtres selon le modèle de Morrone et à la dernière section, nous allons illustrer la performance de ce modèle pour les applications 1D et 2D.

2.2 CARACTÉRISTIQUES D'UN BON DÉTECTEUR D'ARÊTES

2.2.1 Définition des types d'arêtes

Une arête est caractérisée par une discontinuité de l'intensité dans l'image. Une arête peut être de type échelon, ligne ou toit (ou bande de Mach), ayant une certaine orientation, et elle est visible dans une certaine plage d'échelle (figure 2.1).

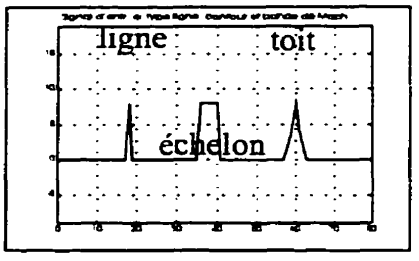


Figure 2.1 Définition des trois types de contours

2.2.2 Propriété d'idempotence

Soit ε, un opérateur de détection d'arêtes défini par:

$$\varepsilon: I \rightarrow \varepsilon(I) \qquad (2.1)$$

où I est l'image originale et

ε (I) est l'image binaire qui contient uniquement des lignes et des points décrivant la position des arêtes. Cette image n'est pas unique et dépend des types d'arêtes que l'opérateur est capable de détecter. Une telle image est appelée une carte d'arêtes.

L'opérateur ε est dit idempotent (projection) si $\varepsilon(\varepsilon(I)) = \varepsilon(I)$. Autrement dit, l'idempotence est la capacité qu'un détecteur redonne la même carte d'arêtes que celle à l'entrée. On peut remarquer facilement que les opérateurs basés sur la technique de différentiation sont incapables de détecter une ligne, et par conséquent, échouent au test d'idempotence. En effet, en appliquant un opérateur différentiel à une arête de type ligne, celui-ci détectera deux arêtes de chaque côté de la bonne.

Notre système visuel est idempotent, car l'image que nous captons par nos yeux ne se dégrade pas d'une carte d'arêtes à une autre. Si notre but est de concevoir un bon détecteur d'arêtes, nous devrons trouver des opérateurs mathématiques qui satisferont au critère d'idempotence. Nous allons voir, dans les prochaines sections, comment de tels opérateurs peuvent être déduits à partir d'expérimentations en neuropsychologie et en psychologie. De plus, nous allons démontrer mathématiquement l'idempotence de ces opérateurs (section 2.4).

2.3 DESCRIPTION DES PHÉNOMÈNES BIOLOGIQUES

Dans cette section, nous allons décrire sommairement les démarches expérimentales effectuées par Morrone et Burr (1989) pour arriver à démontrer qu'il existe vraisemblablement des filtres pairs et impairs dans notre système visuel et qui sont responsables de la détection d'arêtes. À la section 2.3.1, nous allons d'abord définir certains termes généraux que Morrone et Burr utilisent pour décrire leurs expérimentations. Nous allons voir ensuite comment les auteurs ont préparé les stimulus à la section 2.3.2. Enfin, à la section 2.3.3, nous allons présenter les principaux résultats auxquels les auteurs sont arrivés et leurs raisonnements pour conclure qu'il existe deux types de filtres principaux dans notre système visuel.

2.3.1 Définition des termes généraux

Seuil de contraste : C'est le niveau de contraste minimal avec lequel un observateur peut identifier des objets dans une image. Il varie d'une personne à une autre selon les stimulants. Plus le seuil de contraste d'une personne est bas, plus cette personne éprouve de la facilité à détecter un objet.

Sensibilité au contraste: L'inverse du seuil de contraste.

Procédure Quest: C'est une procédure qui consiste à faire varier l'intensité lumineuse pour arriver finalement à un niveau de contraste optimal grâce auquel un individu parvient à identifier, dans 82 % des cas, l'objet particulier qu'il recherche dans une image. Cette procédure détermine donc la sensibilité au contraste d'un individu, pour une image donnée.

Pixel-ligne: C'est l'un des pixels qui constituent la ligne.

Pixel-contour: C'est l'un des pixels qui constituent le contour.

Pixel d'événement : C'est un pixel-ligne ou un pixel-contour.

Piédestal : C'est une composante complémentaire (orthogonale en phase par rapport au signal de test) que l'on ajoute au signal de test.

2.3.2 Génération des arêtes synthétiques pour la simulation

Ayant défini les termes nécessaires pour faciliter les explications qui vont suivre, nous allons maintenant décrire comment les auteurs préparent les stimulus de tests.

Comme stimulus, Morrone et Burr ont généré des contours ou encore deux régions ayant des niveaux de couleurs vraisemblablement différents. Pour cela, ils multiplient une différence de Gaussien DoG(k)/k à chaque harmonique k d'une sommation des termes en cosinus impair (voir équations 2.2). Ils obtiennent ainsi une onde comme illustrée à la figure 2.2.

$$S(x) = C_o + C \sum_{k=1\ \text{impair}}^{256} \cos\left(\left(\frac{2\pi k x}{T}\right) - \phi\right).\frac{DoG(k)}{k} \qquad (2.2a)$$

$$\text{où} \quad DoG(k) = e^{-\frac{k^2}{2\sigma_H^2}} - e^{-\frac{k^2}{2\sigma_L^2}} \quad \text{avec k impair.} \qquad (2.2b)$$

L'équation 2.2a est une forme déguisée de l'équation d'une onde carrée, lorsque les termes de DoG sont égaux à 1. Le but des auteurs consiste donc à produire des ondes ayant l'allure que l'on voit à la figure 2.2.

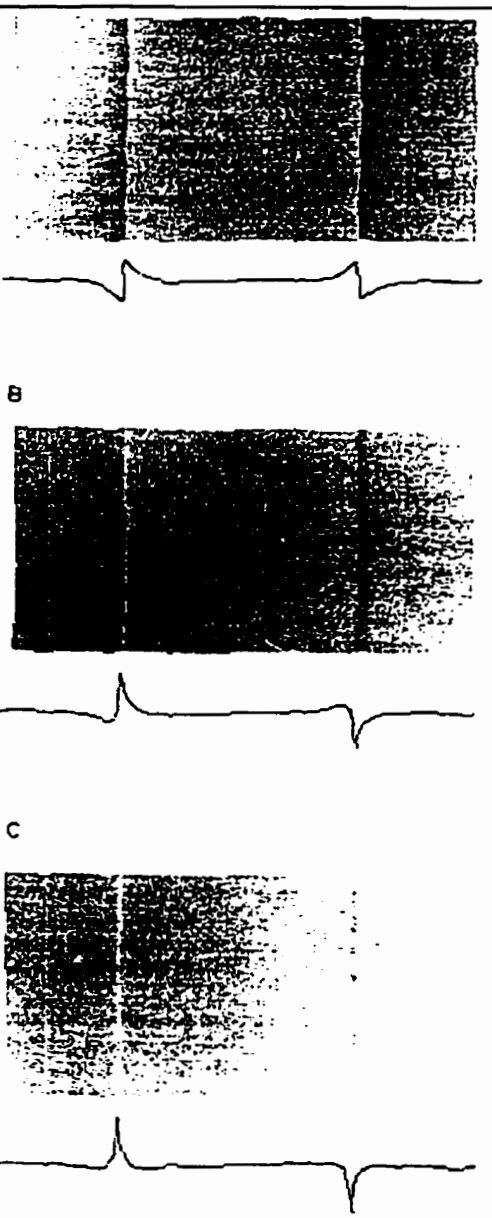


Figure 2.2A *avec Φ=90, il semble y avoir 3 régions (ou deux contours).*
Figure 2.2B *avec Φ=45, il semble y avoir une semi-ligne et un semi-contour.*
Figure 2.2C *avec Φ=0, il semble y avoir 2 lignes (une noire et une blanche).*
Cette figure a été extraite intégralement de l'article de Morrone et al., 1989.

Ces trois figures illustrent trois cas typiques de l'ensemble des ondes formées par l'équation 2.1 en gardant C_0 et C fixes et en faisant varier Φ. Dans les prochaines sections, nous verrons comment les auteurs se servent de ces stimulus pour arriver à

démontrer qu'il existe deux classes de détecteurs, l'un qui agit comme un filtre pair en répondant fortement à la détection d'une ligne et l'autre qui agit comme un filtre impair en répondant fortement à la détection d'un contour.

2.3.3 Expériences

Morrone et Burr font varier la phase Φ de l'équation 2.2a pour qu'elle prenne les valeurs de 0° à 90°. Chaque signal est projeté sur un plan 2D longitudinalement (comme les formes à la figure 2.2). Ils essaient d'ajuster le rapport C_d/C (sensibilité aux contrastes) selon la procédure QUEST pour chercher le niveau de contraste optimal à une valeur Φ donnée et ce, pour deux observateurs pris séparément. Ils obtiennent ainsi les résultats illustrés à la figure 2.3.

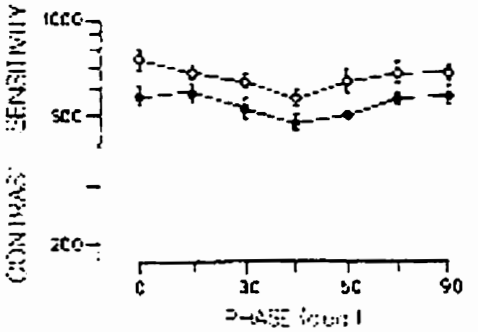


Figure 2.3 ***Courbes de sensibilité aux contrastes en fonction des valeurs Φ de deux observateurs. Cette figure a été extraite intégralement de l'article de Morrone & Burr (1989).***

Pour arriver à dessiner ces courbes, les auteurs de l'article ont mis beaucoup d'efforts afin de s'assurer que la sensibilité mesurée soit optimale. Le résultat nous révèle que l'existence d'un seul détecteur pour toutes les phases Φ est impossible, parce qu'à la valeur de Φ=45°, la sensibilité au contraste a été diminuée, c'est-à-dire qu'il est plus difficile de détecter des arêtes d'un signal ayant la phase fixée à 45°. Il n'existe donc pas de détecteur dédié aux contours du signal avec la phase égalant à 45°. Cette observation

suggère qu'il existe probablement deux détecteurs propres à $\Phi=0^{o}$ et à $\Phi=90^{o}$. Or, on sait que $\Phi=0^{o}$ correspond à l'image ayant deux lignes (figure 2.2c), et que $\Phi=90^{o}$ correspond à l'image ayant deux contours. On peut donc conclure qu'il existe dans notre système visuel deux types de détecteurs dont l'un pour détecter principalement les lignes et l'autre pour les contours.

Pour s'assurer de leurs résultats, les auteurs ont effectué une deuxième série de tests. Cette fois-ci, ils ont ajouté un signal qu'ils ont appelé piédestal dans le signal de test. Les piédestaux sont formés à partir de l'équation 2.2a et 2.2b avec la phase ϕ orthogonale à la phase ϕ du signal de test. Le niveau de contraste du signal de test original est tenu fixe, tandis que le niveau de contraste du piédestal varie de 0 à 0,1. Les signaux résultants sont projetés à l'écran et trois observateurs doivent détecter les contours.

Certains chercheurs (par exemple Nachmias et Sansbury, 1974) montrent que le seuil de contraste d'un piédestal superposé à un signal de test est plus bas que le seuil de contraste du signal de test seul. Ils démontrent que la remarque est vraie pour un certain niveau de contraste du piédestal. Cela implique qu'en général, un ajout du piédestal fait augmenter la sensibilité au contraste chez les observateurs. La figure 2.4 illustre ce principe.

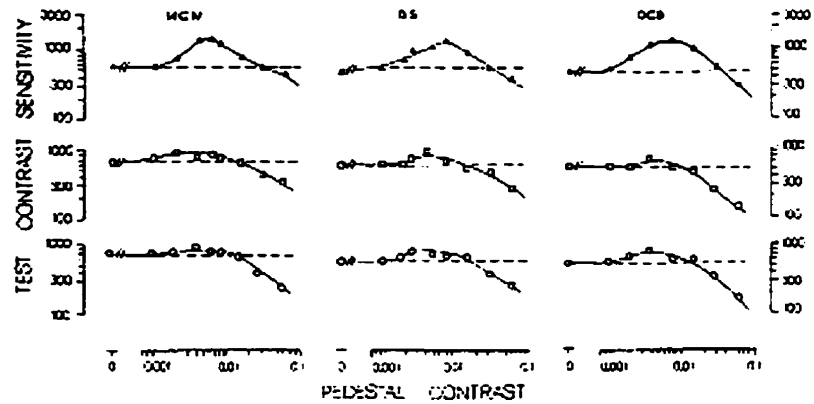


Figure 2.4 ***La sensibilité au contraste de trois observateurs (Morrone, Burr, Spinelli) mesurée en présentation des signaux de piédestal superposés aux signaux de test ϕ=[45°,225°,315°] pour les trois premières courbes, et ϕ=[0°,180°,90°] pour les courbes au milieu et ϕ=[90°,270°,0°] pour les trois courbes les plus basses. Ces mesures sont en fonction du niveau de contraste des piédestaux. Cette figure a été extraite de l'article de Morrone et Burr 1989.***

Dans cette figure, les lignes pointillées représentent les niveaux de sensibilité au contraste en présence des signaux de test purs (sans piédestal). Nous pouvons voir par des lignes pointillées que la sensibilité au contraste, pour les cas où ϕ=[45°,225°,315°], est plus basse que celle des autres cas. Cependant, en présence des piédestaux, la sensibilité au contraste mesurée est plus grande (les arêtes sont plus faciles à détecter), surtout pour les cas où ϕ=[45°,225°,315°]. Ce fait ne contredit pas le résultat de l'expérimentation précédente. La sensibilité au contraste est plus élevée à cause de l'ajout des piédestaux.

Avec cette expérience, les chercheurs ont raisonné comme suit : lorsque le stimulus de test seul ne contient que la composante en sinus (ou cosinus), il devrait être, par conséquent, détecté aisément par le détecteur en sinus (ou cosinus), et lorsqu'on y ajoute un piédestal de phase orthogonale qui ne contient que la composante en cosinus

(ou sinus), ce changement ne devrait pas affecter énormément le résultat précédent. Si un détecteur existait pour les phases autres que 0^o et 90^o (45^o par exemple), il devrait répondre fortement aux deux composantes principales (sinus et cosinus) et permettrait ainsi au piédestal d'influencer le résultat de façon considérable. L'augmentation remarquable de la sensibilité au contraste pour les Φ=[45°,225°,315°], et non pour les autres valeurs de Φ, suggère que toutes les détections d'arêtes sont dictées par deux principaux détecteurs : l'un en sinus et l'autre en cosinus.

Lorsque le niveau de contraste des piédestaux est augmenté, toutes les courbes de sensibilité au contraste sont diminuées. Pour l'instant, Morrone et Burr ne peuvent y trouver aucune explication.

Ces conclusions sont plus évidentes lorsqu'on décompose la source de lumière (l'équation 2.2a) en somme de sinus et de cosinus, selon une propriété trigonométrique très bien connue, pour arriver à l'équation 2.3b. On voit que lorsque $\Phi=90^o$, on ne retrouve plus que la composante en sinus et que lorsque $\Phi=0^o$, on ne retrouve plus que la composante en cosinus. Ainsi, nous pouvons dire que le détecteur responsable de la détection d'une ligne ($\Phi=0^o$) agit comme un filtre pair (forme f(x)=f(-x)) et que le détecteur responsable de la détection d'un contour agit comme un filtre impair (forme f(x)=-f(-x)).

$$S(x) = C_o + C \sum_{k=1\ \text{impair}}^{256} \cos\left(\left(\frac{2\pi k x}{T}\right) - \phi\right).\frac{DoG(k)}{k} \qquad (2.3a)$$

$$= C_o + C \sum_{k=1\ \text{impair}}^{256} \left(\sin(\phi).\sin\left(\frac{2\pi k x}{T}\right) + \cos(\phi).\cos\left(\frac{2\pi k x}{T}\right)\right).\frac{DoG(k)}{k} \qquad (2.3b)$$

$$\text{où} \quad DoG(k) = e^{-\frac{k^2}{2\sigma_H^2}} - e^{-\frac{k^2}{2\sigma_L^2}} \quad \text{avec k impair} \qquad (2.3c)$$

Par conséquent, un filtre pair convolué à l'image d'une ligne donnera une forte réponse à la position où cette ligne (composante en cosinus) apparaît. Par contre, un tel filtre donnera une réponse nulle à l'endroit où un contour existe. Cette propriété est réciproque pour un filtre impair (voir figure 2.5).

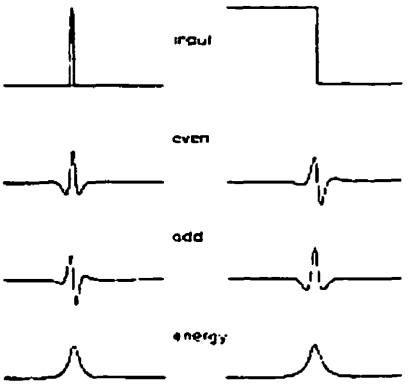


Figure 2.5 ***Réponses des filtres pair et impair à une ligne et à un contour et leur énergie totale. Cette figure a été extraite de l'article de Morrone & Burr (1989).***

Si on suppose que pour un point donné dans l'image, les deux détecteurs pair et impair existent et coopèrent ensemble, alors l'énergie totale associée à ce point sera la racine carrée de la somme des réponses à la sortie de ces deux filtres, chacune élevée au carré. Cette énergie totale nous permet de savoir s'il existe une ligne ou un contour à un endroit donné, lorsque l'énergie dépasse un seuil.

La relation entre l'énergie totale associée à un point et la propriété d'arêtes de ce point peut être démontrée facilement si l'on remarque qu'une ligne ou un contour existent à l'endroit où les harmoniques (en cosinus dans notre exemple) arrivent en phase. Une telle arrivée en phase sous l'effet de la sommation créera des maximums en énergie.

Le modèle se résume donc à calculer l'énergie totale dans la région d'intérêt et les maximums locaux de cette énergie détermineront les positions des pixels d'arête, s'il

y a lieu, dans cette région. Pour comparer la performance de ce modèle versus celle des autres modèles déjà connus, Morrone et Burr (Morrone & Burr, 1988) ont réalisé une expérience utilisant le même type de stimulus que ceux utilisés dans l'expérience ci-dessus. Cependant, les auteurs ont intentionnellement filtré l'onde carrée à travers un filtre passe-bas pour ne garder qu'un signal à trois harmoniques. La projection d'un tel signal en deux dimensions donne une image semblable à celle de la figure 2.6. Avec ce stimulus, nous avons de la difficulté à identifier clairement s'il s'agit, au milieu de l'image, d'une ligne ou d'un contour, car les deux alternent. Quel que soit le résultat de l'observation, aussitôt que l'observateur voit une ligne ou un contour, il doit enregistrer cette position à l'ordinateur grâce à un pixel mobile (contrôlé par l'observateur) superposé à l'image observée. Chacun des deux observateurs marque quarante points. Morrone et Burr calculent ensuite la position moyenne des points marqués et la position moyenne de ces points est confondue aux contours observés. Cependant, ce qui est plus remarquable encore, c'est que cette position tombe parfaitement à la position prévue par le modèle à énergie totale de Morrone. La lettre Z désigne les prédictions de la méthode dans laquelle on détermine les positions de la pente maximale de l'intensité lumineuse (méthode de la première dérivée). La lettre P désigne les prédictions de la méthode dans laquelle on utilise le point d'intensité lumineuse maximale comme point d'arête. La lettre C désigne les prédictions de la méthode dans laquelle on détermine le centre de masse de la « masse lumineuse » de gauche à droite.

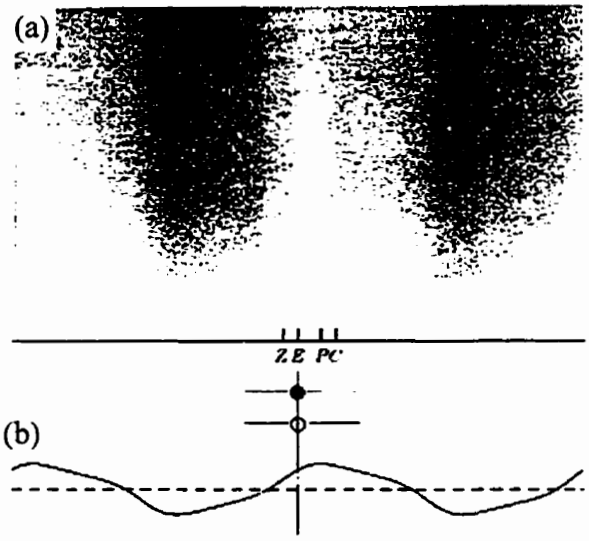


Figure 2.6 ***La performance du modèle à énergie locale versus les autres modèles. La partie (b) illustre le signal 1D généré à l'aide de l'équation 2.1, filtré avec un filtre passe-bas. La partie (a) illustre le même signal projeté sur un plan 2D longitudinalement. La ligne marquée par Z,E,P,C indique les lieux théoriques de détection d'arêtes tels que prédits par les différentes méthodes de détection. Le cercle noir représente la position moyenne où le premier observateur voit une arête. Le cercle blanc représente la position moyenne où le deuxième observateur voit une arête. Ces figures ont été extraites partiellement de l'article de Morrone et Burr (1988).***

On peut donc conclure que notre système visuel agit comme s'il était composé de deux filtres : l'un pair et l'autre impair. Ces filtres convoluent avec l'entrée I (intensité lumineuse d'une rangée de pixels). Les sorties des deux filtres sont combinées selon la formule d'énergie pour obtenir un signal dont les maximums locaux identifieront des lignes et des contours. Pour cela, un système de seuil est nécessaire. La valeur optimale de ce seuil reste à déterminer. Plus le seuil est élevé, plus les lignes et les contours seront imperceptibles. Par contre, plus un seuil est bas, plus on risque de détecter des lignes ou des contours artificiels. En posant l'hypothèse qu'on peut arriver à déterminer un seuil optimal, on peut désormais prétendre détecter tous les pixels-lignes ou des pixels-contours (voir figure 2.7).

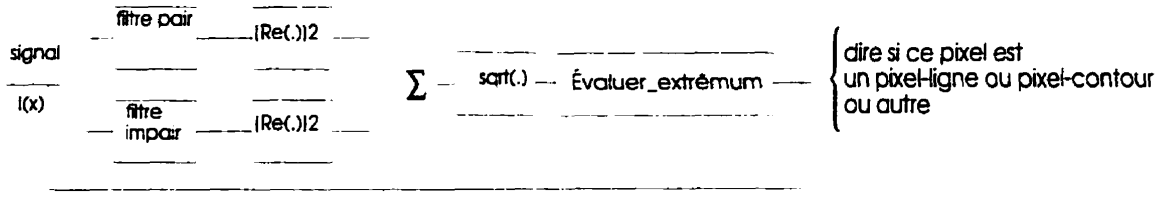


Figure 2.7 Modèle simplifié de détection des lignes ou des contours

Il est à noter qu'un tel modèle ne tient pas compte de la connexité des pixels-ligne ou des pixels-contour d'une ligne ou d'un contour en particulier.

Les expériences précédentes ont été réalisées sur une base plutôt psychologique. Du côté biologique, D. Marr (1982) a fait des découvertes menant à des conclusions semblables en examinant les réponses du cerveau de singes à différents stimulus. La figure 2.8 tirée partiellement du livre de D. Marr (1982) montre les réponses enregistrées en électrophysiologie pour des stimulus de type échelon brusque, de type ligne et de type échelon normal.

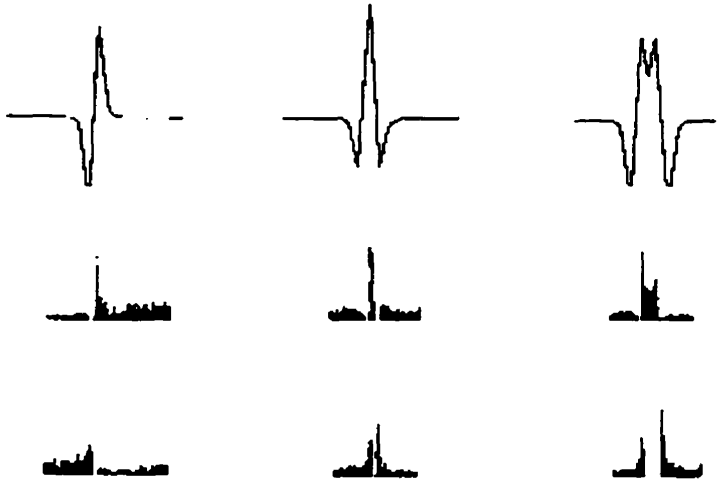


Figure 2.8 Les réponses enregistrées en électrophysiologie reflètent également les propriétés symétriques paire et impaire. Ces figures sont extraites en partie du livre de D.Marr (1982).

La première ligne de cette figure représente l'intensité des trois stimulus utilisés dans l'expérience. Les lignes 2 et 3 illustrent les réponses électrophysiologiques enregistrées. On voit donc que les réponses à une arête de type échelon ont des formes symétriques impaires et les réponses face à une arête de type ligne ont des formes symétriques paires. Cette observation confirme le fondement du modèle de Morrone. Le seul doute qui reste encore est que nous ne sommes pas sûrs s'il existe d'autres classes de détecteur de forme non symétrique qui interviennent dans les processus de détection d'arêtes (Morrone et Burr, 1989).

2.4 PROPRIÉTÉS MATHÉMATIQUES DES DÉTECTEURS DU MODÈLE DE MORRONE

Nous venons de voir la preuve de l'existence de deux types de détecteurs dans notre système visuel par la voie psychologique et biologique. Cependant, est-il vrai que tous les couples de fonctions paire et impaire peuvent être utilisés pour calculer l'énergie totale d'un signal d'entrée afin de déduire les points d'arêtes ? C'est ce que nous allons vérifier dans cette section. En fait, nous allons illustrer le propos de Christian Ronse (1993) en ce qui concerne les conditions nécessaires et suffisantes à respecter avec les filtres implantés selon le modèle de Morrone pour que ce modèle soit le plus efficace possible.

Supposons que l'on a deux opérateurs linéaires , ψ_e et ψ_o, et que ψ_e possède une forte réponse à une arête de type ligne et une faible réponse à un échelon et ψ_o agit de façon contraire. On peut remarquer que, dans une image I, les arêtes de type ligne et échelon apparaissent comme valeurs crêtes de signal $(\psi_e (I)^2 + \psi_o(I)^2)^{1/2}$. Ce signal est appelé la fonction d'énergie associée à l'image I. Plusieurs choix d'opérateurs sont possibles ; par exemple, le produit de convolution de I par deux fonctions F_e et F_o respectivement paire et impaire est un choix valable. La fonction d'énergie devient alors

$E_n(x)= [(I\otimes F_e)^2+(I\otimes F_o)^2]^{1/2}$. L'avantage de restreindre les opérateurs à un produit de convolution permet de mieux contrôler l'échelle spatiale, en sélectionnant les largeurs de bande de F_e et F_o.

Ces détecteurs peuvent détecter avec beaucoup de précision toutes formes d'arêtes, notamment les lignes, les contours et les bandes de Mach. Ils sont idempotents et stables à différentes échelles, comparativement aux détecteurs basés sur la différentiation (proposé par Marr-Hildreth, 1980). Toutefois, ils ont un défaut qu'on ne trouve pas chez les autres types de détecteurs: ils sont sensibles au décalage de niveau de gris (« sensitivity to grey-level shift », ainsi indentifié par Christian Ronse, 1993).

2.4.1 Problème avec le décalage de niveau de gris et la correction

Soit une image originale I et un niveau de gris g, (I+g) est l'image résultante. On a donc (I+g)(p)=I(p)+g à tout point p de I. Comme les deux images représentent la même scène, elles possèdent donc la même carte d'arêtes (edge map) ε (I)= ε (I+g). L'addition d'un niveau de gris à l'image ne devrait poser aucun problème pour les opérateurs basés sur la différentiation, puisque la constance disparaît une fois que le signal résultant est dérivé. Cependant, ce n'est pas toujours le cas avec des détecteurs d'énergie. En effet, les opérateurs basés sur le produit de convolution sont invariants si $\int_s F_e(x)dx=0$. Pour mieux illustrer ce résultat, Christian Ronse démontre ce qui suit:

soit $\int_s F_e(x)dx =m \neq 0$; et nous avons par définition

$$E_n(x)^2 = ((I+g)\otimes F_e)^2 + ((I+g)\otimes F_o)^2 \qquad (2.4)$$

$$= (I\otimes F_e + g\int_s F_e(x)dx)^2 + (I\otimes F_o + g\int_s F_o(x)dx)^2 \qquad (2.5)$$

$$= (I \otimes F_e + gm)^2 + (I \otimes F_o + 0)^2 \quad (2.6)$$

$$= (I \otimes F_e)^2 + (I \otimes F_o)^2 + \mathbf{2gm(I \otimes F_e)} + (gm)^2 \quad (2.7)$$

Comme $(gm)^2$ est une constante, elle n'influence donc pas la position des maximums de $E_n(x)^2$. Cependant pour $m \neq 0$, $2gm(I \otimes F_e)$ n'est pas un terme constant. Si I est un signal pair et contient un échelon au point x=0; le terme $2gm(I \otimes F_e)$ correspondra alors à un produit de deux fonctions paire et impaire. De plus, le résultat sera une fonction impaire, et par conséquent, sa dérivée au point zéro ne sera pas nulle. Le terme $2gm(I \otimes F_e)$ ne sera pas maximal à x=0. Les formules 2.6 et 2.7 ne peuvent pas être maximales à x=0. Autrement dit, pour $m = \int_S F_e(x)dx \neq 0$, le détecteur d'énergie localisera un échelon à une position incorrecte. Par conséquent, en choisissant une F_e telle que $\int_S F_e(x)dx = m = 0$, nous pouvons éliminer le terme $\mathbf{I \otimes F_e}$ indésirable, pour assurer une invariance au décalage de niveau de gris.

L'un des détecteurs d'énergie les plus connus est celui de Gabor (R. Owens & al., 1989). Pour une $\sigma>0$ et une fréquence f, la fonction de Gabor est définie par:

$$G_\sigma^f = \frac{1}{\sigma\sqrt{2\pi}} \exp(\frac{-x^2}{2\sigma^2}) \exp(2\pi \cdot if \cdot x) \quad (2.8)$$

La partie réelle de G(x) s'appelle la fonction cosinus de Gabor, notée $GC(\sigma,f)$ (paire). La partie imaginaire de G(x) s'appelle la fonction sinus de Gabor, notée $GS(\sigma,f)$ (impaire).

Puisque $\int_R GC_{(\sigma,f)}(x)dx \neq 0$, le détecteur localisera donc un échelon à une position incorrecte. Des corrections mineures aux fonctions de Gabor assurent toutefois l'invariance.

$$F_e(x) = G_\sigma(x)\left[\cos(2\pi fx) - e^{-2(\pi\sigma f)^2}\right] \quad (2.9)$$

$$F_o(x) = G_\sigma(x)\left[\sin(2\pi fx) - e^{-2(\pi\sigma f)^2} 2\pi fx\right] \quad (2.10)$$

Les termes ajoutés garantissent que $\int_s F_e(x)dx = \int_s xF_o(x)dx=0$ (dualité avec le fait que par définition $\int_s F_o(x)dx = \int_s xF_e(x)dx=0$ [Christian Ronse, 1993]).

2.4.2 Élaboration du modèle d'énergie de Morrone

Nous avons vu précédemment que les positions des pixels d'événements correspondent à des endroits où il y a des maximums locaux en énergie. Pour continuer, nous allons, dans cette sous-section, définir ce que Christian Ronse (1993) entend par les arrivées en phase. Nous profitons également de l'occasion pour faire un petit rappel de la propriété de la transformée de Fourier et des opérateurs mathématiques qui lui sont associés.

Si $F \in L^1$, où L^1 est l'espace des fonctions intégrables qui convergent, alors sa transformée de Fourier notée $F^\wedge$ est:

$$\mathbf{F^\wedge(\nu) = \int_{\Re} F(x)\ \exp(-i2\pi\nu x)\ dx} \quad (2.11)$$

où $F^\wedge(0)=\int_{\Re} F(x)dx$ est le niveau dc de F. (2.12)

Selon la formule de la transformée de Fourier, on obtient :

$$\mathbf{F(x)=\int_{\Re} F^\wedge(\nu)\ \exp(i2\pi\nu x)\ d\nu} \quad (2.13)$$

Si $F \in \Re$ ($\Re$ l'ensemble des réels), on a la propriété suivante:

$$\mathbf{F^{\wedge}(\nu) = F^{\wedge}(-\nu)}. \qquad (2.14)$$

Sous une autre forme, (2.11) peut être récrit comme suit :

$$F^{\wedge}(\nu) = F^{A}(\nu)\exp(i\,F^{\phi}(\nu)) \qquad (2.15)$$

où $F^{A}(\nu)$ et $F^{\phi}(\nu)$ est l'amplitude et la phase de $F^{\wedge}(\nu)$ respectivement.

$$\text{D'après (2.14)} \Rightarrow F^{A}(\nu)\exp(-i\,F^{\phi}(\nu)) = F^{A}(-\nu)\exp(i\,F^{\phi}(-\nu)). \qquad (2.16)$$

Par identité, on a: $F^{A}(-\nu) = F^{A}(\nu)$, $F^{\phi}(-\nu) = -F^{\phi}(\nu)$

$\Rightarrow F^{A}(\nu)$ est paire et $F^{\phi}(\nu)$ est impaire. L'expression (2.13) devient alors:

$$F(x) = \int_{\Re} F^{A}(\nu)\cos(2\pi\nu x + F^{\phi}(\nu))d\nu \qquad (2.17)$$

$$\mathbf{\Rightarrow F(x) = 2\int_{\Re_+} F^{A}(\nu)\cos(2\pi\nu x + F^{\phi}(\nu))\,d\nu} \qquad (2.18)$$

La transformée de la translatée de F est:

$$[\tau_{-p}(F)]^{\wedge}(\nu) = \exp(2\pi i p\nu)F^{\wedge}(\nu). \qquad (2.19)$$

Elle a le même module que F, mais sa phase est avancée proportionnellement à $2\pi p\nu$.

$$\Rightarrow [\tau_{-p}(F)]^{\phi}(\nu) = F^{\phi}(\nu) + 2\pi p\nu. \qquad (2.20)$$

Selon la définition de Christian Ronse (1993), dans les signaux 1-D, tout point p, dont les phases $F^{\phi}(\nu) + 2\pi p\nu$ (pour toutes fréquences positives) sont proches les une des autres de façon maximale, est appelé le point d'arrivé en phase maximale (MPC ou « maximum phase congruency »). Dans la section suivante, nous allons démontrer

mathématiquement pourquoi les points correspondant aux arrivées en phase correspondent aux points d'arête.

2.4.3 Formules mathématiques du modèle de Morrone

Assumons le signal: $I: \Re \rightarrow \Re$ ($I \in L^1$ ou L^2), où L^2 est l'espace des fonctions intégrables qui convergent. Le modèle de Morrone consiste à appliquer au signal I deux filtres linéaires et invariants par translation F_e et F_o tels que:

$$(F_e\hat{})^A = (F_o\hat{})^A = \mathbf{F} \tag{2.21}$$

$$\text{et } (F_e\hat{})^\phi = 0,\ (F_o\hat{})^\phi = \pi/2. \tag{2.22}$$

La somme des carrés des signaux de sortie donne le carré de la fonction d'énergie $E_n(x)^2$ et le modèle de Morrone prétend que les arêtes dans l'image sont localisées aux maximums locaux de $E_n(x)$.

Preuve de Chirstian Ronse 1993

D'après le modèle de Morrone, on a (en domaine fréquentielle) :

$$\psi_e(I)^A(\nu) = \psi_o(I)^A(\nu) = I^A(\nu)\ F_e(\nu) = I^A(\nu)\ F_O(\nu) = \mathbf{I}^A(\nu)\ \mathbf{F}(\nu) \tag{2.23}$$

$$\text{et } \psi_e(I)^\phi(\nu) = I^\phi(\nu),\ \ \psi_o(I)^\phi(\nu) = I^\phi(\nu) + \pi/2 \tag{2.24}$$

On applique l'équation (2.18) et utilisant $\cos(\theta+\pi/2) = -\sin(\theta)$. On a donc $\forall x \in \Re$

$$\psi_e(I)(x) = 2\int_{\Re+} I^A(\nu) F(\nu) \cos(2\pi\nu x + I^\phi(\nu))\, d\nu \tag{2.25}$$

$$\text{et } \psi_{o\,(I)}(x) = 2\int_{\Re+} I^{A}(\nu)F(\nu)\sin(2\pi\nu x + I^{\phi}(\nu))\,d\nu \qquad (2.26)$$

$$\text{Or, } E_n(x)^2 = \psi_e(I)(x)^2 + \psi_o(I)(x)^2 \qquad (2.27)$$

En utilisant (2.25) et (2.26) et à quelques manipulations trigonométriques près, on obtient:

$$E_n(x)^2 = 4\iint_{\Re 2+} I^{A}(\nu)F(\nu)I^{A}(\mu)F(\mu)$$

$$\cos[(2\pi\nu x + I^{\phi}(\nu)) - (2\pi\mu x + I^{\phi}(\mu))]\,d\mu d\nu. \qquad (2.28)$$

$E_n(x)^2$ est maximal lorsque l'argument du cosinus tend vers zéro. Pour cela, les phases de I à x pour toute fréquence ν devraient être égales entre elles. Le point x est donc un point d'arrivée en phase ou MPC, et le terme MPC est équivalent au terme de d'énergie local maximale.

Par conséquent, pour satisfaire la condition en ce qui concerne l'arrivée en phase, le modèle de Morrone requiert la transformée de Hilbert suivante:

$$F_o = H(F_e) \text{ car } \forall\, F \in L^2 \quad H(F)\hat{}(\nu) = \pm\, i(F)\hat{}(\nu) \qquad (2.29)$$

Aussi, les contraintes mathématiques sur les deux filtres pair et impair du modèle de Morrone doivent être les suivantes:

- Fo(x), filtre impair, fonction réelle bornée,
- Fe(x), filtre pair, fonction réelle bornée à composante dc nulle,
- Fo^(f)=iFe^(f), pour tout f>0.

Ces contraintes assurent deux caractéristiques importantes d'un bon détecteur : notamment, l'invariance à l'augmentation graduelle du niveau de gris et le respect du principe idempotent.

2.5 INTERPRÉTATION DU SIGNAL D'ÉNERGIE

La théorie du modèle de Morrone prédit que les extremums locaux du signal d'énergie correspondent aux positions de changement d'intensité (position des pixels d'événement). Il faut donc avoir une idée claire de ce que l'on veut dire par extremum local. Comme le niveau maximal d'amplitude du signal d'énergie varie d'un signal d'entrée à un autre, les niveaux d'amplitude des extremums locaux ne peuvent pas être prédéterminés. Le seul moyen d'y arriver est de trouver un seuil variable. Ainsi, lorsque l'énergie d'un point dépasse ce seuil, étant donné que les niveaux d'énergie de ses deux voisins gauches et droites sont inférieurs ou égaux à son énergie, nous pouvons conclure qu'un tel point correspond à une arête.

Le seuil variable doit être évidemment fonction du niveau maximal d'amplitude (globale) du signal d'énergie. Dans notre cas, nous avons fixé ce seuil à :

$$\text{seuil} = \frac{\text{Max(Énergie)}}{3} \tag{2.30}$$

Un tel seuil est approprié pour un signal d'entrée binaire (deux niveaux d'amplitude). Pour le cas des images en couleurs, ce seuil doit être plus bas et le choix le plus approprié est très dépendant des attentes des usagers. La prochaine section est réservée à la discussion de la performance du modèle de Morrone en utilisant un tel seuil appliqué à des signaux 1D et des images en couleurs.

2.6 PERFORMANCE DU DÉTECTEUR DE CONTOURS DE MORRONE

2.6.1 Performance mesurée avec les signaux 1D

Nous allons, dans cette section, discuter de la performance du modèle de Morrone appliqué à des signaux 1D. Nous allons voir que ce modèle nous permet de détecter les arêtes avec beaucoup plus de précision qu'avec la méthode de passage par zéro de la deuxième dérivée de Marr et Hildreth 1980, qui était connue auparavant comme le modèle qui modélisait le mieux les filtres de détection d'arêtes chez l'être humain.

Supposons que nous avons un signal 1D idéal tel qu'illustré à la figure 2.9. Le signal contient trois types d'événements principaux : arêtes de type ligne, arête de type échelon et arête de type toit. (figure 2.9a). En dérivant ce signal deux fois, nous pouvons détecter la présence des événements en examinant les points de passage par zéro dans ce signal dérivé (figure 2.9b) selon le principe de Marr et Hildreth (1980). La figure 2.9c illustre le signal d'énergie du signal d'entrée après avoir filtré avec un filtre pair et un filtre impair ayant les caractéristiques suivantes :

$$Fe = \frac{1}{\sigma f} \bullet e^{-\frac{t^2}{2\sigma^2}} \bullet \left(\cos(2\pi f t) - e^{-2\bullet(\pi\sigma f)^2} \right)$$

$Fo = Im\{Hilbert(Fe)\}$

avec

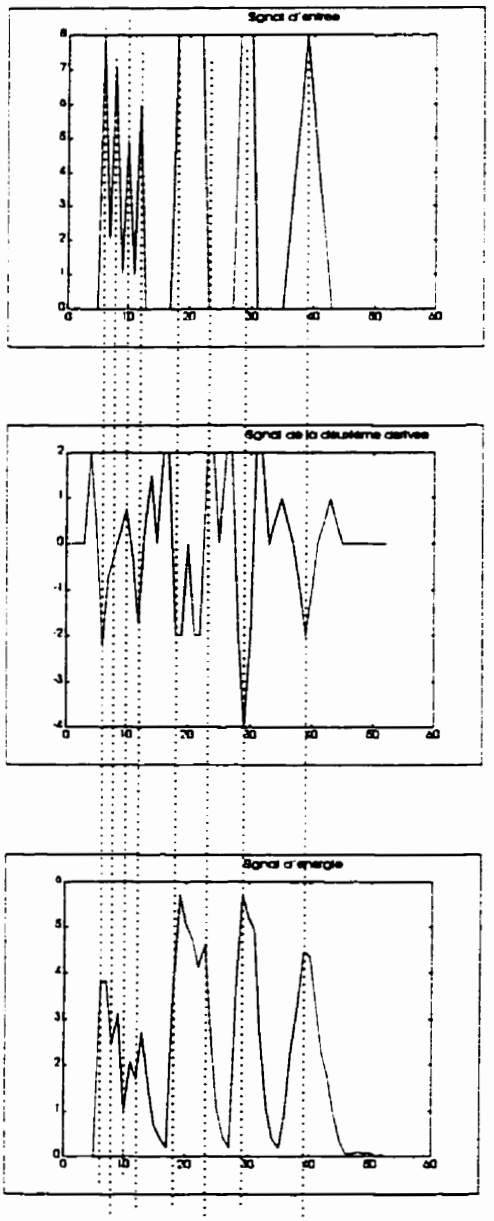
$f = 0{,}02$

$\sigma = 1{,}2$

$t = [0:1:49]$

On peut démontrer facilement que le filtre pair Fe possède une valeur moyenne presque nulle, ce qui permet de corriger le défaut des filtres par rapport au décalage du niveau de gris. De plus, la transformée de Hilbert appliquée au filtre Fe donne un filtre impair Fo qui est exactement orthogonal au filtre Fe. Ce faisant, nous obtenons deux filtres idéaux, car ces deux filtres satisfont aux contraintes mathématiques discutées à la section 2.4.3.

La figure 2.9c illustre les résultats de détection d'arêtes utilisant la méthode de la deuxième dérivée et la méthode proposée selon le modèle de Morrone. Les trois petites figures sont alignées pour que l'on puisse mieux comparer la performance entre les deux méthodes.



(a)

(b)

(c)

Figure 2.9 ***a)Le signal d'entrée en 1D composé de 4 pixels-ligne, un contour de largeur de 5 pixels, un contour de largeur de 3 pixels et un contour de type de toit. La deuxième dérivée du signal est présentée à la figure 2.8b. Les passages par zéro de ce signal devraient correspondre à des événements d'arêtes. Le signal d'énergie est présenté à la figure 2.8c. Les lignes pointillées correspondent à des pixels d'événements marqués dans le signal d'entrée.***

On constate aisément que la méthode utilisant le modèle de Morrone donne un signal résultant beaucoup plus facile à interpréter. Les événements de lignes sont détectés avec une marge d'erreur minimale. En effet, lorsque le seuil utilisé pour le signal d'énergie est le tiers de son énergie maximale, tous les quatre pixels-lignes seront perceptibles avec un décalage d'un pixel. Par contre, la méthode de la deuxième dérivée ne fait pas ressortir de façon évidente les pixels-lignes. Pour ce qui est des arêtes de type échelon et de type toit, nous avons constaté que la méthode utilisant le modèle de Morrone est nettement plus performante que la méthode de la deuxième dérivée. En plus de ces caractéristiques intéressantes, on constate également que nos filtres sont capables d'amincir les contours de largeur inférieure à quatre pixels (voir figure 2.9c).

Il n'y a pas de règle absolue pour déterminer la fréquence f, l'écart-type σ et le pas d'incrémentation en t pour arriver à concevoir des filtres idéaux. La méthode que l'on a utilisée pour déterminer ces valeurs est la méthode d'essai-erreur. Cependant, ces paramètres sont choisis de façon à ce qu'ils soient près des valeurs physiques mesurées dans notre système visuel (D. Marr, 1982).

En effet, nous avons d'abord fixé σ égal à 1,2, tandis que le pas d'incrémentation en t est égal à 1. Nous faisons varier f graduellement en examinant constamment la performance de nos détections. Nous avons enfin trouvé une valeur intéressante pour f égale à 0,02. Le filtre Fe ainsi créé possède la forme d'un filtre passe-bande que l'on retrouve d'ailleurs en partie dans l'article de Morrone et Burr (1988) (figure 2.10).

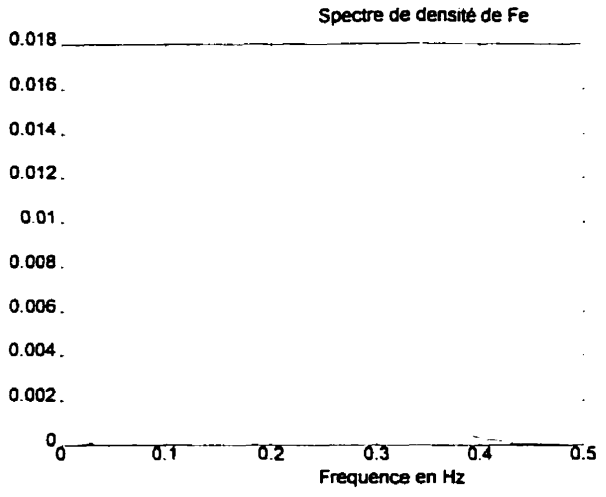


Figure 2.10 ***Le filtre Fe en domaine fréquentiel pour f>0***

Bien que le modèle de Morrone soit un outil performant et précis, nous détectons plus ou moins bien le nombre d'événements qui existent dans un signal lorsque nous n'utilisons pas un niveau de seuil approprié. Dans le cas de signaux ou d'images à deux niveaux seulement, ce seuil se trouve souvent au milieu de la valeur maximale du signal d'énergie (résultant). Cependant, pour un signal à niveaux multiples ou pour une image de couleurs variées, il nous faut des seuils multiples. Dans la prochaine section, nous allons présenter la performance de la méthode utilisant le modèle de Morrone appliqué à une image 2D en noir et blanc.

2.6.2 Performance mesurée avec les signaux 2D (images)

Notre description serait incomplète si nous discutions de la performance du modèle de Morrone sans montrer ce que ce modèle peut détecter dans une image 2D. Dans la figure 2.11a, nous présentons le visage d'une personne en noir et blanc. Les contours dans cette image ont été détectées par un outil de détection d'arêtes (Sobel) fourni avec CorelPHOTO-PAINT© (figure 2.11b). Finalement, la figure 2.11c montre les arêtes que les filtres de Morrones trouvent.

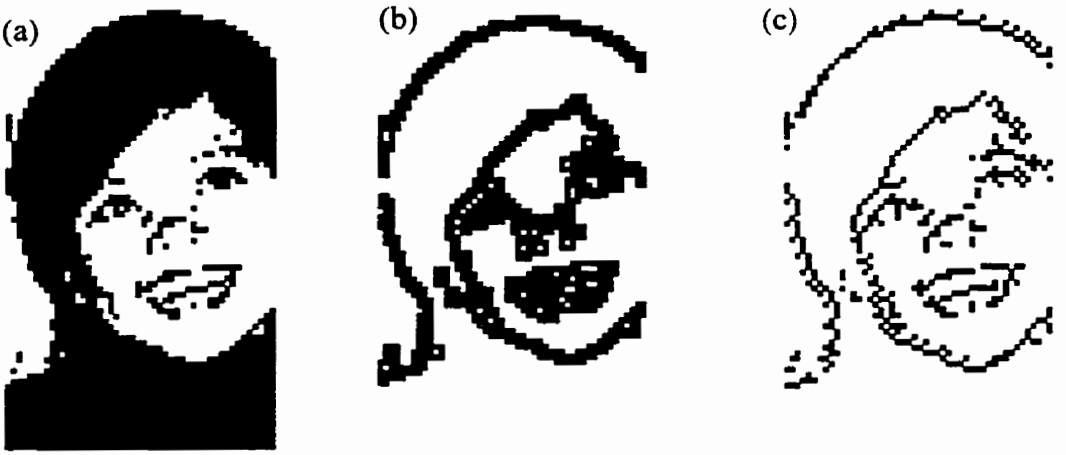


Figure 2.11a ***L'image originale du visage d'une personne.***

Figure 2.11b ***Illustre le résultat de détection d'un opérateur commercial.***

Figure 2.11c ***Illustre le résultat de détection selon le modèle de Morrone.***

En observant ces images, nous pouvons conclure que le modèle de Morrone est nettement plus performant que le modèle utilisé à la figure 2.11b. Pour comprendre l'importance de la propriété d'idempotence, nous avons appliqué les filtres de Morrone une deuxième fois à l'image originale. La figure 2.12a montre le résultat de détection d'arêtes de l'image de la figure 2.11b par le même outil, c'est-à-dire CorelPHOTO-PAINT©, tandis que la figure 2.12b montre le résultat de détection d'arêtes en utilisant le modèle de Morrone appliqué une deuxième fois à l'image de la figure 2.11c.

Figure 2.12a ***L'image originale du visage d'une personne passée deux fois à travers l'opérateur commercial.***

Figure 2.12b ***Illustre le résultat de détection d'arêtes selon le modèle de Morrone appliqué deux fois consécutives à l'image originale.***

Les filtres de Morrone semblent donc très appropriés pour le traitement des images binaires contenant beaucoup de lignes droites, parce que ces filtres détectent très bien les lignes droites de largeurs de quatre pixels et moins (en plus de pouvoir effectuer des amincissements). C'est la raison pour laquelle nous avons décidé d'utiliser ces filtres dans notre projet pour détecter les arêtes, car les images de cartes géographiques sont constituées surtout de lignes droites.

2.7 CONCLUSION

Nous venons de voir clairement la relation entre le modèle d'énergie de Morrone et son lien biologique. Malgré que ce modèle prenne plus de temps de calcul que les autres modèle (parce que les filtres ne peuvent être mis en cascade), il mérite d'être utilisé et analysé plus profondément, car il modélise mieux le caractère physiologique de notre système visuel. Pour l'instant, ce qui nous motive à l'utiliser, c'est que ce modèle répond très bien à notre besoin, grâce à ses propriétés d'idempotence et d'amincissement.

Chapitre 3

La transformée de Hough

Caractéristiques et améliorations

3.1 OBJECTIF

Nous avons vu, dans le chapitre précédent, que le modèle de Morrone nous permet d'identifier tous les pixels de type contour dans une image. La prochaine étape consiste à faire des analyses sur cet ensemble de points pour y déduire les pixels de type coin. Puisque les coins sont formés à partir de segments de lignes ayant des orientations très variées et à cause du bruit de numérisation, nous ne pouvons utiliser les méthodes de type première dérivée ou dérivée seconde (E.R Davies, 1990). Il existe également d'autres méthodes pour détecter les coins comme, par exemple, la méthode utilisant les filtres à base médiane ou la méthode des pyramides hiéarchiques de Rosenfeld (1992), qui sont valables pour des applications particulières. Cependant, le fondement

mathématique de ces méthodes n'est pas inspiré par la vision humaine, ce qui va à l'opposé de notre objectif initial de design.

Or, certains chercheurs (par exemple D. Marr, 1982) ont remarqué que les êtres humains tentent toujours d'abord de détecter les lignes dans une image. Ils estiment ensuite la position des coins par les rencontres de ces lignes. En effet, même avec une image qui n'est composée que par un certain nombre de pixels noirs éparpillés, nous avons tendance à essayer, par tous les moyens, de trouver une ou des interprétations possibles à ces arrangements. L'exemple suivant illustre une image simple avec quelques pixels noirs (figure 3.1).

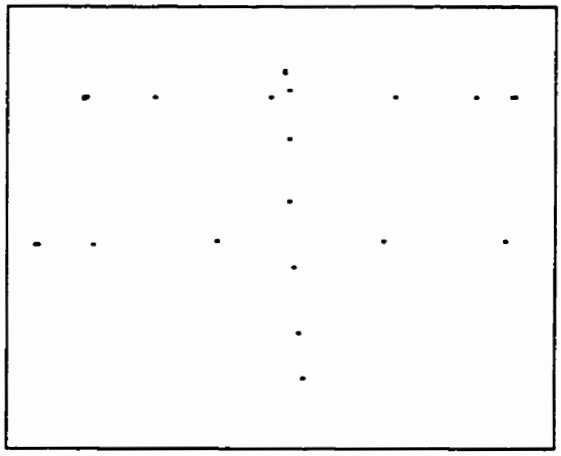


Figure 3.1 ***Une loi de vision chez l'être humain : la continuité spatiale ou un des principes de Gestalt (D. Marr, 1982).***

En observant cet exemple, nous voyons immédiatement qu'il s'agit d'une image composée de trois lignes. Ce comportement automatique chez l'être humain a été étudié par une école allemand et correspond au principe de bonne continuité ou de continuité spatiale, selon le terme utilisé par D. Marr (1982). Notre travail de détection s'arrête habituellement à cette étape et nous ne détectons généralement pas des points

d'intersections, à moins que cette opération n'ait été spécifiée comme étant un but de l'observation.

Pour rechercher les points d'intersection, l'être humain commene toujours par localiser les lignes possibles. Il déduit ensuite la position des points d'intersections. Conscient de ce phénomène, nous nous sommes demandés, dans cette étude, si nous pouvions concevoir des algorithmes qui suivraient les mêmes étapes qu'un être humain pour arriver à estimer le nombre de lignes principales et le nombre de coins principaux qui existent dans l'image.

Ce raisonnement nous a amenés à utiliser un outil mathématique exceptionnel en matière de regroupement des pixels en lignes ou en courbes paramétrisables : la transformée de Hough (Hough 1962). Cet opérateur a été développé dans les années 1970 par M. Hough, pour détecter uniquement des lignes droites dans une image. Plus tard, vers les années 1980, d'autres scientifiques (par exemple, Muller 1987) ont apporté une extension à l'algorithme pour qu'il soit capable de paramétriser même les courbes circulaires et elliptiques. Ces nouveaux algorithmes portent ainsi le nom de « transformée de Hough généralisée ».

Puisqu'une image peut être décrite entièrement par des segments droits, les chercheurs ont beaucoup apprécié et valorisé la transformée de Hough, dès son apparition. Cependant, malgré la solidité de son fondement mathématique, cette méthode semble encore imparfaite pour certaines applications réelles, car les paramètres de lignes qu'elle propose contiennent encore des imprécisions.

Dans la prochaine section, nous allons d'ailleurs voir une description mathématique de la transformée de Hough. Ensuite, nous parlerons sommairement des techniques que les scientifiques utilisent en général pour implanter cette transformée de Hough et des faiblesses de ces techniques d'implantation. Puis, nous décrirons les

améliorations que présente notre nouvelle méthode d'implantation. Enfin, à la dernière section, nous présenterons des résultats d'exécution selon la méthode que nous avons proposée afin de mieux évaluer la performance des changements.

3.2 RAPPEL DU CONCEPT DE LA TRANSFORMÉE DE HOUGH

3.2.1 Énoncé du théorème de la transformée de Hough

Tout pixel (x_i,y_i) appartenant à une droite faisant l'angle θ avec l'axe horizontal donnera la même valeur d par rapport à l'origine, lorsqu'on utilise la formule suivante:

$$\mathbf{d=x_i \sin(\theta) + y_i \cos(\theta)} \qquad (3.1)$$

3.2.2 Démonstration

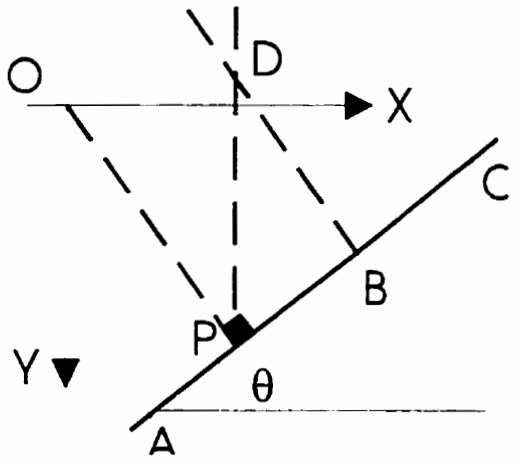


Figure 3.2 Illustration du théorème et démonstration de la transformée de Hough

Soit une droite AC dans le plan cartésien d'origine O, telle qu'illustrée à la figure 3.2.

Soit le point P, ayant la plus courte distance avec l'origine O.

$\Rightarrow OP \perp AC$

Alors, nous avons

$$d = |OP| \tag{3.2}$$

$$= d(\sin^2(\theta)+\cos^2(\theta)) \tag{3.3}$$

$$= d\sin(\theta)\sin(\theta)+d\cos(\theta)\cos(\theta) \tag{3.4}$$

De plus, soit un point B quelconque sur AC.

Si de P, on dessine une ligne verticale, et de B on dessine une ligne perpendiculaire à AC, alors ces deux lignes se croiseront au point D. Suite à ces constructions, nous déduisons les relations suivantes:

$$\theta + \angle DPB = 90° = \angle DPB + \angle OPD \tag{3.5}$$

$$\Rightarrow \angle OPD = \theta \tag{3.6}$$

$$\text{D'où} \quad d\sin(\theta) = P_x \quad \text{et} \quad d\cos(\theta) = P_y \tag{3.7}$$

$$(3.4) \Rightarrow d = |OP| = P_x\sin(\theta)+P_y\cos(\theta) \tag{3.8}$$

De plus,

$$B_x = P_x + |PB|\cos(\theta) \tag{3.9}$$

$$B_y = P_y - |PB|\sin(\theta) \tag{3.10}$$

$$=>B_x \sin(\theta) + B_y \cos(\theta) = (P_x+|PB|\cos(\theta))\sin(\theta)+(P_y - |PB|\sin(\theta))\cos(\theta) \quad (3.11)$$

$$(3.8)=>B_x \sin(\theta) + B_y \cos(\theta) = P_x \sin(\theta)+P_y \cos(\theta) = d \quad (3.12)$$

Si la droite AC est orientée avec un angle $\theta > 90°$, alors

$$B_x = P_x + |PB| \cos(180°-\theta) \quad (3.13)$$

$$B_y = P_y + |PB| \sin(180°-\theta) \quad (3.14)$$

Donc,

$$B_x = P_x - |PB| \cos(\theta) \quad (3.15)$$

$$B_y = P_y + |PB| \sin(\theta) \quad (3.16)$$

Ainsi, les termes $|PB|\cos(\theta)\sin(\theta)$ disparaîtront dans tous les cas.

Finalement, nous obtenons la relation suivante:

$$B_x \sin(\theta) + B_y \cos(\theta) = P_x \sin(\theta)+P_y \cos(\theta) = d, \quad (3.17)$$

ce qui complète la démonstration de l'équation 3.1.

3.2.3 Propriétés de la transformée de Hough

La transformée de Hough, implantée selon l'équation (3.1) va générer trois types de paramètre d :

a)d=0 si la droite correspondant passe par l'origine O,

b)d>0 si $y > -x\tan(\theta)$ pour $90>\theta>0$ OU $y < -x\tan(\theta)$ pour $180>\theta>90$,

c)d<0 sinon, $y < -x\tan(\theta)$ pour $90>\theta>0$ OU $y > -x\tan(\theta)$ pour $180>\theta>90$.

Nous verrons, plus loin dans ce chapitre, comment nous effectuons une translation d'axe pour que les coordonnées graphiques du point (i,j) soient exprimées en (i',j') (par rapport à une origine virtuelle) avant d'effectuer la transformée de Hough sur ce point. Ce faisant, nous serons assurés que :

$$-i_{limite}<i'<i_{limite} \quad \text{et} \quad -j_{limite}<j'<j_{limite} \tag{3.18}$$

Les valeurs limites de i et de j auront des grandeurs finies. Comme nous verrons dans le chapitre 4, nous ferons en sorte que $i_{limite}=j_{limite}=35$.

Autrement dit, les transformées de Hough seront appliquées à des points (i',j') dans le scénario suivant :

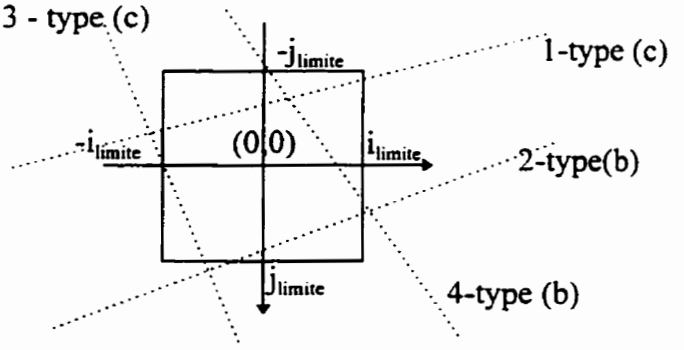


Figure 3.3 ***Scénario pour les calculs de la transformée de Hough***

On peut vérifier facilement que la droite 1 et la droite 3 sont de type c, tandis que la droite 4 est de type b.

3.3 LES MÉTHODES D'IMPLANTATION CONVENTIONNELLES ET LEURS LIMITES

3.3.1 La tendance actuelle des méthodes d'implantation

La plupart des scientifiques utilisent la transformée de Hough pour extraire de façon approximative les deux paramètres (d, θ), car il n'apparaît pas critique pour eux que soient introduites des erreurs d'arrondi au niveau de la déduction de ces paramètres, s'il n'est pas nécessaire de calculer les points d'intersection entre les éléments. Dans cette optique, ils suivent généralement les étapes qui sont décrites ci-dessous.

D'abord, les scientifiques construisent un tableau à deux dimensions MxN où M représente le nombre de valeurs possibles de d que l'on peut extraire dans l'image et N représente le nombre de valeurs possibles de θ. Il est évident que le domaine de θ est toujours bien défini [0°,360°] ou [0°,180°] si l'on convertit tous les angles en des angles positifs. La dimension N est donc très bien déterminée par :

$$N = \frac{360^o}{\Delta\theta} \qquad \text{ou} \qquad N = \frac{180^o}{\Delta\theta} \tag{3.19}$$

où Δθ représente le pas de quantification pour le paramètre θ.

La dimension M, quant à elle, doit couvrir une zone assez large, car selon l'équation (3.1), d peut être positif ou négatif (voir section 3.23). Toutefois, on sait que d ne peut pas être plus grand que la largeur ou la hauteur de la zone d'analyse (rectangulaire a priori). Nous sommes donc certains que :

$$-\max(L,H) <= d <= \max(L,H) \tag{3.20a}$$

où L est la largeur (en pixels) de la zone d'image,

et H est la hauteur (en pixels) de la zone d'image.

La dimension M peut donc être déterminée en divisant cet intervalle par le pas de quantification pour d :

$$M = 2*\max(L,H)/\Delta d \qquad (3.20b)$$

Les valeurs Δd et $\Delta\theta$ devront être choisies de façon optimale selon les analyses de Ming Zhang (1996) ou de W. Niblack et D. Petkovie (1990) ou encore de Chee-Woo Kang et al., (1991).

Chaque casier du tableau contient un compteur entier qui est mis à zéro initialement. La tendance générale consiste à extraire les paramètres (d, θ) reliés à chaque pixel d'événement. Pour chaque (d, θ) trouvé, on incrémente de 1 le compteur du casier correspondant. La méthode utilisée pour remplir ce tableau consiste donc à choisir une série de valeurs de θ représentées par le tableau. Pour chaque θ donné, on calcule d à l'aide de la formule 3.1. Cette valeur d ne correspond pas toujours à une des valeurs d définie dans le tableau. Dans ce cas, il faut incrémenter le compteur du casier ayant une valeur d la plus proche de la valeur calculée. On appelle cette action une vote. Évidemment, on commet une certaine erreur en procédant ainsi. Plus une droite contient de pixels, plus le compteur correspondant à cette droite (caractérisée par d et θ) sera élevé. Ainsi, à la fin de cette analyse, on peut estimer le nombre de lignes ainsi que leur position dans la zone d'analyse en comparant les compteurs à un seuil quelconque.

3.3.2 Les erreurs causées par les méthodes conventionnelles d'implantation

À cause du bruit et des discontinuités, les pixels d'événement formant une droite ne donnent pas toujours les mêmes d et θ, de telle sorte que les casiers adjacents au casier auront aussi des valeurs de compteurs très élevées.

Les défauts de ces méthodes d'implantation sont dus à l'erreur d'arrondi de la valeur d, à l'erreur de quantification de θ, à la grande taille de la mémoire requise et au manque d'information sur la longueur de la droite (l'information sur les deux points d'extrémités). De plus, il existe une autre source d'erreur qui fait que la méthode de Hough ne réussit pas complètement son travail de vote : celle-ci réside dans la façon même d'implanter les formules qui mesurent la similarité entre deux couples (d_1,θ_1) et (d_2,θ_2). En effet, on a toujours pensé que c'était seulement dans le cas où deux couples avaient les mêmes d et les mêmes θ que ces deux couples représentaient la mêmes droite. En réalité, il peut exister des couples ayant des d totalement différentes (en termes de signe et de grandeur), mais qui représentent le même segment. Dans ce cas, au lieu d'incrémenter la valeur de compteur du casier (d_1,θ_1), on incrémente celle du casier (d_2,θ_2), duquel on ne considérera jamais équivalent à (d_1,θ_1). Cela veut donc dire qu'il y a une forte possibilité que l'accumulateur de Hough contienne encore plusieurs casiers redondants après l'étape de construction de l'accumulateur de Hough.

Ayant identifié ce problème, nous avons pensé à un moyen pour y remédier. Dans les prochaines sections, nous allons discuter de notre nouvelle méthode d'implantation.

3.4 UNE NOUVELLE MÉTHODE D'IMPLANTATION

D'abord, il faut rappeler que notre objectif consiste à détecter les points de croisement entre les droites qui sont identifiées par leur valeur de compteur élevée dans l'accumulateur (ou tableau) de Hough. La position de croisement (x_c,y_c) entre deux droites (d_1,θ_1) et (d_2,θ_2) est déterminée de la manière suivante :

$$x_c\sin(\theta_1) + y_c\cos(\theta_1)=d_1 \tag{3.21}$$

$$x_c\sin(\theta_2) + y_c\cos(\theta_2)=d_2 \tag{3.22}$$

Ces deux équations utilisent le même point (x_c,y_c), car ce dernier est le point d'intersection des deux droites. Pour trouver ce point, il faut résoudre le système d'équations ci-dessus. Sous forme de matrice, nous avons :

$$\begin{bmatrix} \sin(\theta_1) & \cos(\theta_1) \\ \sin(\theta_2) & \cos(\theta_2) \end{bmatrix} \begin{bmatrix} x_c \\ y_c \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \end{bmatrix} \tag{3.23}$$

$$x_c = \frac{d_1 * \cos(\theta_2) \; - \; d_2 * \cos(\theta_1)}{\Delta} \tag{3.24}$$

$$y_c = \frac{d_2 * \sin(\theta_1) \; - \; d_1 * \sin(\theta_2)}{\Delta} \tag{3.25}$$

$$\text{avec } \Delta = \sin(\theta_1 - \theta_2) \tag{3.26}$$

Lorsque les deux droites sont parallèles, les deux valeurs de θ sont presque égales, de telle sorte que $\Delta\approx 0$. Dans ce cas, on ne peut pas calculer x_c et y_c, puisque deux lignes parallèles ne se croisent jamais. Aussi, ces dernières formules n'utilisent pas l'information sur le point de commencement ni de fin des droites. Elles permettent donc

de résoudre un des défauts de la méthode de Hough. Le défaut concernant l'erreur de l'arrondissement des valeurs de d et le défaut concernant l'erreur de la quantification de valeurs de θ de la méthode discutée précédemment pourront entraîner des erreurs très graves à la détermination précise du point de croisement de deux droites.

Afin de minimiser les possibilités d'erreur de quantification tout en diminuant l'espace de mémoire de l'accumulateur, nous avons décidé de ne pas quantifier à l'avance les valeurs de θ. Nous déduisons plutôt les valeurs de θ au fur et à mesure et nous formerons un nouveau casier, si nécessaire, pour y emmagasiner exactement cette valeur de θ. Ainsi, les θ qui définissent l'accumulateur ont des valeurs « surmesurées ». Par conséquent, les valeurs de d correspondantes sont aussi précises. Étant donné que le triplet (d, θ,compteur) est toujours liés ensemble, nous avons créé à la place un tableau à une dimension, de façon à ce que chaque élément du tableau contienne trois champs respectivement. La détermination de (d, θ) est effectuée en considérant à tour de rôle chaque pixel d'événement comme le pixel central. À partir de ce point, nous calculons les valeurs θ que ce pixel central forme avec les pixels voisins. Pour chaque θ, nous déterminons d à l'aide de la formule 3.1, en utilisant les coordonnées du pixel central comme valeur du couple (x,y). Il faut ensuite vérifier si le couple (d, θ) existe déjà dans le tableau. Si oui, nous incrémentons simplement le compteur du triplet (d_H,θ_H, compteur) correspondant et remettons à jour (d_H, θ_H) par les formules de calcul de la moyenne.

$$d_H = \frac{d_H * \text{compteur} + d}{\text{compteur} + 1} \tag{3.27}$$

$$\theta_H = \frac{\theta_H * \text{compteur} + \theta}{\text{compteur} + 1} \tag{3.28}$$

$$\text{compteur} = \text{compteur} + 1 \tag{3.29}$$

Ces formules assurent la convergence des paramètres, car en moyenne les (d_H, θ_H) devraient se converger vers des valeurs fixes s'ils représentent des droites. Lorsque nous ne pouvons pas trouver, dans le tableau de Hough, le triplet dont les paramètres ressemblent à (d, θ), nous ajoutons alors ce nouveau couple (d, θ) au tableau en forçant le compteur correspondant à 1.

La vérification de l'existence de (d, θ) dans le tableau est effectuée à l'aide d'une formule de mesure de similarité. Cette formule suppose que deux segments formant une même droite doivent avoir nécessairement $d_H \approx d$ et $\theta_H \approx \theta$.

Cette action de regroupement nous permet non seulement de nous assurer de la convergence des valeurs (d, θ), mais elle nous permet également d'obtenir un tableau de grandeur réduite ne décrivant que les droites les plus importantes. Cette dernière propriété nous permet de réduire le temps de calcul à l'étape de détermination des points de croisement, là où nous devons considérer les éléments du tableau de Hough deux à deux.

Dans toutes les méthodes, nous devons, à un certain moment, comparer les compteurs avec un seuil pour décider si l'élément correspondant mérite d'être traité. Un accumulateur de grandeur réduite ne décrivant que les lignes les plus importantes contiendra des valeurs de compteur beaucoup plus significatives, ce qui simplifiera énormément la tâche de comparaison et d'élimination.

Malgré ces belles formules, l'accumulateur de Hough contient encore des casiers redondants lorsqu'on constate qu' il existe aussi des cas où nous avons les mêmes θ, mais des d très différents. Pourtant, ces deux segments appartiennent à une seule droite. Un exemple d'une telle situation est illustré à la figure 3.4.

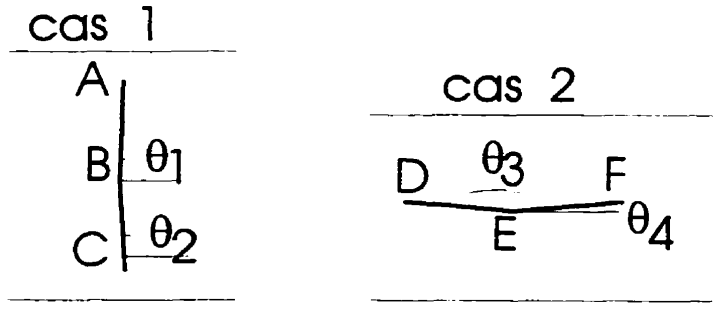


Figure 3.4 Cas de regroupement

Supposons que les angles $\theta_1 \approx \theta_2$ et $\theta_3 \approx \theta_4$, de telle sorte que l'on puisse dire que :

ABC est une droite verticale,

DEF est une droite horizontale,

D'après (3.1),

$$d_{AB}=x_B \sin(\theta 1) + y_B \cos(\theta 1) \tag{3.30}$$

$$d_{BC}=x_B \sin(\theta 2) + y_B \cos(\theta 2) \tag{3.31}$$

$$d_{DE}=x_E \sin(\theta 3) + y_E \cos(\theta 3) \tag{3.32}$$

$$d_{EF}=x_E \sin(\theta 4) + y_E \cos(\theta 4) \tag{3.33}$$

$$|d_{AB} - d_{BC}| = x_B (\sin(\theta 1) - \sin(\theta 2)) + y_B(\cos(\theta 1) - \cos(\theta 2)) \tag{3.34}$$

$$|d_{DE} - d_{EF}| = x_E (\sin(\theta 3) - \sin(\theta 4)) + y_E(\cos(\theta 3) - \cos(\theta 4)) \tag{3.34}$$

$$\sin(\theta 1) - \sin(\theta 2) \approx 0 \tag{3.35}$$

$$\sin(\theta 3) - \sin(\theta 4) \approx 0 \tag{3.36}$$

Alors,

$$|d_{AB}-d_{BC}| \approx y_B(\cos(\theta 1) - \cos(\theta 2)) \tag{3.37}$$

$$|d_{DE}-d_{EF}| \approx y_E(\cos(\theta 3) - \cos(\theta 4)) \tag{3.38}$$

Si y_B ou y_E sont très grands, alors $|d_{AB}-d_{BC}|$ et $|d_{DE}-d_{EF}|$ seront aussi très grands. Dans ce cas, le test de similarité échoue, car on arrive à conclure que AB =/= BC et DE =/= EF, ce qui est faux.

Ayant découvert cette source d'erreur, nous devons trouver une solution pour garder la variable y mimime pour regrouper efficacement les triplets. Pour cela, il suffit de noter que, par convention (chapitre 1), on sait a priori la position graphique approximative de l'objet que l'on veut chercher dans l'image. Si on se sert de cette position comme étant une nouvelle origine (origine virtuelle) pour déterminer les (d_H, θ_H), et qu'on fait tous les autres calculs par rapport à cette origine virtuelle, on peut éliminer une source d'erreur importante dans la méthode d'implantation usuelle de Hough.

Dans notre application, nous pouvons même déterminer les points de croisements des lignes par rapport à l'origine virtuelle et exprimer ensuite la position de ce point par rapport à l'origine globale (0,0). Avec les autres applications, lorsque le but est d'exprimer les paramètres des lignes par rapport à l'origine globale, nous pouvons toujours utiliser l'approche suivante (voir figure 3.5).

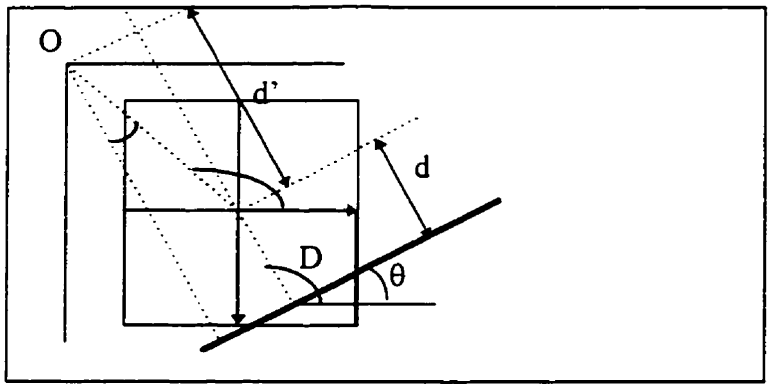


Figure 3.5 Expression de (d_H, θ_H) par rapport à l'origine globale O

Supposons que l'on a une droite correspondant à une distance d et un angle θ par rapport à une origine virtuelle V et l'axe horizontal respectivement. On voit que l'angle O constitue la différence entre l'angle V et l'angle D. Ainsi, la distance d' peut être déduite directement par

$$d' = \cos(|\angle V - \angle D|) \bullet \sqrt{(x_V - x_O)^2 + (y_V - y_O)^2} \tag{3.39}$$

$$= \cos(|\angle V - \angle D|) \bullet \sqrt{(x_V)^2 + (y_V)^2} \tag{3.40}$$

$$\text{où } \angle D = \theta + 90° \text{ et } \angle V = \tan^{-1}\left(\frac{-y_V}{x_V}\right) \tag{3.41}$$

Alors, la distance entre la droite et l'origine (0,0) est d'+d.

3.5 RÉSULTATS D'APPLICATION

Nous avons implanté les deux méthodes de détection de droites : la méthode classique décrite à la section 3.3.1 et la nouvelle méthode proposée à la section 3.4. Pour

comparer les performances entre le cas où nous ne faisons pas de translation d'axe et le cas où nous utilisons une origine virtuelle, nous avons utilisé une image synthétique. Une telle image (illustrée à la figure 3.6) ne contient presque pas de pixels de bruits (pixels qui ne sont associés à aucune ligne).

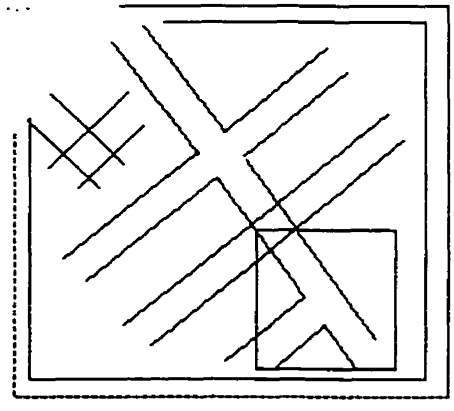


Figure 3.6 Image synthétique pour les tests utilisant les deux méthodes de détection de ligne

Les deux algorithmes ont été exécutés pour des points à l'intérieur de la zone carrée située dans la partie inférieure droite de l'image. À l'œil nu, nous pouvons voir qu'il y existe cinq lignes principales selon deux orientations principales : trois lignes à 37° et deux à 124°. À cause d'un des défauts mineurs de notre méthode d'implantation, nous obtiendrons inévitablement des lignes secondaires ayant un champ compteur très négligeable par rapport au champ compteur des lignes principales. Ce défaut, qui sera discuté en détail dans le prochaine chapitre, s'applique également aux deux méthodes d'implantation et n'influence donc pas le résultat de notre comparaison.

3.5.1 Le tableau de Hough selon la méthode conventionnelle (sans translation d'axe)

Nous présentons ici le résultat d'exécution de l'algorithme de détection de lignes selon la méthode conventionnelle d'implantation. Les rangées en caractères gras désignent les lignes qui peuvent être considérées comme lignes principales. On peut y voir facilement que les lignes ayant des angles près d'un des angles principaux ont leur champ compteur généralement plus élevé que les champs compteurs des autres éléments détectés par la méthode de Hough.

d(1) = 182,752903, Theta(1) = 37,834719, count(1) = 13
d(2) = -65,259297, Theta(2) = 156,801409, count(2) = 1
d(3) = -85,977535, Theta(3) = 162,394902, count(3) = 3
d(4) = -104,951919, Theta(4) = 169,613873, count(4) = 2
d(5) = 128,000000, Theta(5) = 0,000000, count(5) = 2
d(6) = -1,122640, Theta(6) = 135,314193, count(6) = 14
d(7) = 74,692487, Theta(7) = 114,416094, count(7) = 6
d(8) = 35,991493, Theta(8) = 125,346648, count(8) = 87
d(9) = 41,190698, Theta(9) = 126,869898, count(9) = 43
d(10) = 53,234437, Theta(10) = 123,635273, count(10) = 72
d(11) = -27,062044, Theta(11) = 143,463157, count(11) = 4
d(12) = -35,545596, Theta(12) = 146,001997, count(12) = 6
d(13) = -56,796127, Theta(13) = 153,434949, count(13) = 2
d(14) = -71,484231, Theta(14) = 157,500000, count(14) = 2
d(15) = -93,618751, Theta(15) = 165,963757, count(15) = 2
d(16) = 64,301503, Theta(16) = 120,589251, count(16) = 15
d(17) = 29,956066, Theta(17) = 127,037130, count(17) = 51
d(18) = -21,256612, Theta(18) = 141,053751, count(18) = 4
d(19) = 184,958931, Theta(19) = 45,000000, count(19) = 7
d(20) = -81,902991, Theta(20) = 161,565051, count(20) = 1
d(21) = -15,064121, Theta(21) = 139,663947, count(21) = 3
d(22) = -43,560641, Theta(22) = 149,036243, count(22) = 1
d(23) = 182,818162, Theta(23) = 56,148448, count(23) = 6
d(24) = 13,679159, Theta(24) = 130,477994, count(24) = 19
d(25) = 18,116156, Theta(25) = 128,659808, count(25) = 26
d(26) = 24,877936, Theta(26) = 127,061251, count(26) = 39
d(27) = 46,087430, Theta(27) = 124,897329, count(27) = 50
d(28) = 60,690669, Theta(28) = 118,397845, count(28) = 12
d(29) = 172,240076, Theta(29) = 67,732864, count(29) = 3
d(30) = 159,709709, Theta(30) = 75,963757, count(30) = 2
d(31) = 176,649370, Theta(31) = 63,434949, count(31) = 1
d(32) = 165,034799, Theta(32) = 72,809828, count(32) = 2

d(33) = 133,000000, Theta(33) = 0,000000, count(33) = 2
d(34) = 152,233462, Theta(34) = 80,537678, count(34) = 1
d(35) = 132,333333, Theta(35) = 90,000000, count(35) = 3
d(36) = 96,401812, Theta(36) = 106,138863, count(36) = 3
d(37) = -97,984393, Theta(37) = 165,963757, count(37) = 1
d(38) = 110,293305, Theta(38) = 100,386127, count(38) = 2
d(39) = 103,873400, Theta(39) = 102,673088, count(39) = 2
d(40) = 83,800358, Theta(40) = 108,434949, count(40) = 2
d(41) = -75,949393, Theta(41) = 158,198591, count(41) = 1
d(42) = -61,491869, Theta(42) = 153,434949, count(42) = 2
d(43) = -114,834141, Theta(43) = 171,869898, count(43) = 1
d(44) = 90,339559, Theta(44) = 107,190172, count(44) = 4
d(45) = -47,959688, Theta(45) = 146,749283, count(45) = 2
d(46) = 68,140265, Theta(46) = 114,881821, count(46) = 2
d(47) = -54,006660, Theta(47) = 147,624995, count(47) = 3
d(48) = -40,630532, Theta(48) = 144,720017, count(48) = 2
d(49) = 49,127599, Theta(49) = 120,692895, count(49) = 9
d(50) = 9,045197, Theta(50) = 133,743227, count(50) = 7
d(51) = 79,040594, Theta(51) = 114,183230, count(51) = 4
d(52) = -4,949747, Theta(52) = 135,000000, count(52) = 1
d(53) = 120,490996, Theta(53) = 98,130102, count(53) = 1
d(54) = 144,333333, Theta(54) = 90,000000, count(54) = 3
d(55) = 178,739208, Theta(55) = 75,009180, count(55) = 2
d(56) = 209,012945, Theta(56) = 52,074672, count(56) = 5
d(57) = 211,799605, Theta(57) = 37,287524, count(57) = 45
d(58) = 117,380877, Theta(58) = 99,462322, count(58) = 1
d(59) = 186,890610, Theta(59) = 71,565051, count(59) = 1
d(60) = 163,765931, Theta(60) = 81,869898, count(60) = 1
d(61) = 166,864972, Theta(61) = 80,537678, count(61) = 1
d(62) = 191,080503, Theta(62) = 68,198591, count(62) = 1
d(63) = 194,163848, Theta(63) = 18,434949, count(63) = 1
d(64) = 197,668409, Theta(64) = 63,434949, count(64) = 1
d(65) = 204,823827, Theta(65) = 26,565051, count(65) = 1
d(66) = 201,308374, Theta(66) = 60,255119, count(66) = 1
d(67) = 211,778481, Theta(67) = 45,000000, count(67) = 10
d(68) = 208,885277, Theta(68) = 30,963757, count(68) = 1
d(69) = 170,000000, Theta(69) = 0,000000, count(69) = 5
d(70) = -112,260857, Theta(70) = 161,565051, count(70) = 2
d(71) = 83,181729, Theta(71) = 116,565051, count(71) = 1
d(72) = 228,815335, Theta(72) = 45,000000, count(72) = 32
d(73) = 229,718672, Theta(73) = 37,513590, count(73) = 53
d(74) = 203,244854, Theta(74) = 14,036243, count(74) = 1
d(75) = 197,881180, Theta(75) = 11,309932, count(75) = 1
d(76) = -127,331203, Theta(76) = 165,963757, count(76) = 1
d(77) = 225,692138, Theta(77) = 30,963757, count(77) = 1
d(78) = 211,398262, Theta(78) = 18,434949, count(78) = 2
d(79) = 216,520764, Theta(79) = 21,801409, count(79) = 1
d(80) = 194,812800, Theta(80) = 9,462322, count(80) = 1
d(81) = -9,545942, Theta(81) = 135,000000, count(81) = 8
d(82) = -100,275483, Theta(82) = 158,198591, count(82) = 1

d(83) = -143,355917, Theta(83) = 170,537678, count(83) = 1
d(84) = -66,026955, Theta(84) = 149,036243, count(84) = 1
d(85) = -148,633845, Theta(85) = 171,869898, count(85) = 1
d(86) = 5,656854, Theta(86) = 135,000000, count(86) = 1
d(87) = -96,510228, Theta(87) = 156,801409, count(87) = 1

La taille maximale prévue pour l'accumulateur de Hough avait été préalablement fixée à 140, mais grâce à la « propreté » de l'image, nous n'avons pas atteint cette limite lors du remplissage. Si cela s'était produit, nous aurions eu à réorganiser le tableau de Hough avant de pouvoir continuer l'analyse. Nous parlerons d'ailleurs de cet aspect au prochain chapitre.

Les éléments qui pourraient être associés à la première ligne principale à 37° (la ligne la plus haute parmi les trois lignes à 37°) sont :

d(1) = 182,752903, Theta(1) = 37,834719, count(1) = 13
d(19) = 184,958931, Theta(19) = 45,000000, count(19) = 7
d(23) = 182,818162, Theta(23) = 56,148448, count(23) = 6

Les éléments qui pourraient être associés à la deuxième ligne principale à 37° sont :

d(56) = 209,012945, Theta(56) = 52,074672, count(56) = 5
d(57) = 211,799605, Theta(57) = 37,287524, count(57) = 45
d(67) = 211,778481, Theta(67) = 45,000000, count(67) = 10

Les éléments qui pourraient être associés à la dernière ligne principale à 37° sont :

d(72) = 228,815335, Theta(72) = 45,000000, count(72) = 32
d(73) = 229,718672, Theta(73) = 37,513590, count(73) = 53

Les éléments qui pourraient être associés à des lignes principales à 124° sont :

d(8) = 35,991493, Theta(8) = 125,346648, count(8) = 87
d(9) = 41,190698, Theta(9) = 126,869898, count(9) = 43
d(10) = 53,234437, Theta(10) = 123,635273, count(10) = 72
d(16) = 64,301503, Theta(16) = 120,589251, count(16) = 15
d(17) = 29,956066, Theta(17) = 127,037130, count(17) = 51
d(24) = 13,679159, Theta(24) = 130,477994, count(24) = 19
d(25) = 18,116156, Theta(25) = 128,659808, count(25) = 26
d(26) = 24,877936, Theta(26) = 127,061251, count(26) = 39
d(27) = 46,087430, Theta(27) = 124,897329, count(27) = 50

Nous ne pouvons séparer facilement les éléments formant les lignes à 124° en deux groupes distinctes, car ce n'est pas évident. Prenons par exemple les éléments 8, 9 et 10 de la dernière liste. En calculant la différence entre les d et les angles θ, nous arrivons à la conclusion que les éléments 8 et 9 pourraient se confondre en une seule droite, parallèle à la droite représentée par l'élément 10. En effet,

d(8) = 35,991493, Theta(8) = 125,346648, count(8) = 87
d(9) = 41,190698, Theta(9) = 126,869898, count(9) = 43
d(10) = 53,234437, Theta(10) = 123,635273, count(10) = 72

$$\text{nous avons } |d_8-d_9|=6 < |d_9-d_{10}|=12 \qquad (3.42)$$

$$\text{et } |\theta_8-\theta_9|\approx 1 < |\theta_8-\theta_{10}|\approx 2 < |\theta_9-\theta_{10}|\approx 3 \qquad (3.43)$$

Cependant, si l'on essaie de reconstruire ces droites en faisant varier x de 0 à la largeur de l'image et si l'on applique la formule inverse de 3.1 pour retrouver y, nous obtenons les trois courbes illustrées à la figure 3.7.

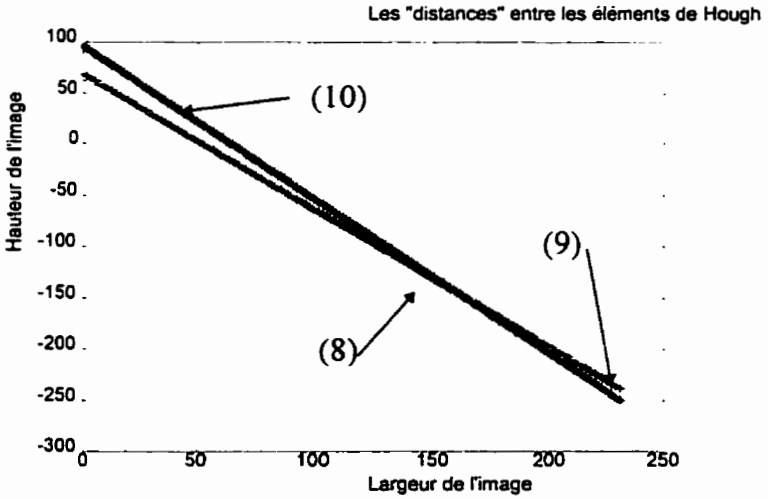


Figure 3.7 Comparaison de distance entre trois éléments de Hough. Cas de deux droites principales parallèles sans translation d'axe

Comme on peut le voir sur cette figure, les deux droites correspondant aux éléments 9 et 10 semblent former une seule droite, tandis que l'élément 8 semble

parallèle à 10 et 9. Pour être plus précis, nous avons fait un agrandissement de l'image dans la zone d'analyse, c'est-à-dire la zone qui entoure l'intersection de rues recherchée. Cette fois-ci, la figure 3.8 suggère que ce sont plutôt les droites 9 et 10 qui devraient être regroupées.

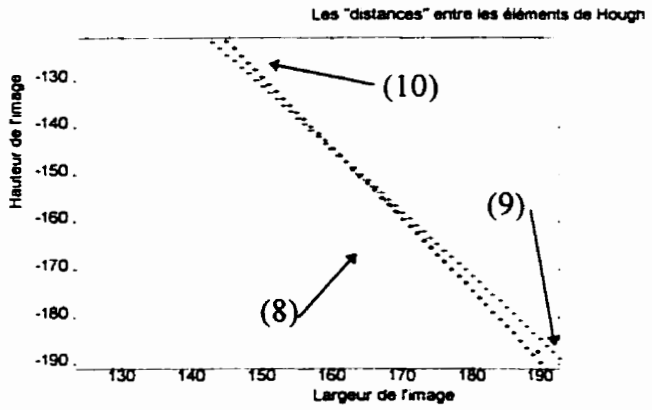


Figure 3.8 La partie d'agrandissement de la figure 3.7 pour couvrir la zone d'analyse seulement

Par conséquent, on peut conclure que les formules qui mesurent la similarité entre les éléments de Hough, telles qu'illustrées précédemment, peuvent conduire à de graves erreurs de regroupement.

3.5.2 Le tableau de Hough selon la méthode modifiée (avec translation d'axe)

Nous venons de voir qu'il est impossible de se contenter d'utiliser simplement la formule de différence pour regrouper les éléments de Hough. Il faut donc appliquer la nouvelle méthode d'implantation. Ce qui suit est le résultat d'exécution de la méthode de Hough implantée selon la méthode proposée à la section 3.4. Les rangées en caractères gras désignent les lignes qui peuvent être considérées comme lignes principales. On peut y voir facilement que les lignes ayant des angles près d'un des angles principaux ont leur

champ compteur généralement plus élevé que les champs compteurs des autres éléments détectés par la méthode de Hough.

d(1) = -1,547848, Theta(1) = 135,000000, count(1) = 15
d(2) = -38,673665, Theta(2) = 38,760131, count(2) = 5
d(3) = 3,663654, Theta(3) = 142,179155, count(3) = 11
d(4) = -38,486812, Theta(4) = 45,000000, count(4) = 7
d(5) = 8,087708, Theta(5) = 148,659385, count(5) = 4
d(6) = 13,416408, Theta(6) = 153,434949, count(6) = 1
d(7) = 18,875945, Theta(7) = 168,121870, count(7) = 4
d(8) = 9,596799, Theta(8) = 126,983749, count(8) = 138
d(9) = 10,404571, Theta(9) = 135,000000, count(9) = 7
d(10) = -6,663309, Theta(10) = 126,764818, count(10) = 122
d(11) = 9,256154, Theta(11) = 122,463228, count(11) = 40
d(12) = -36,166221, Theta(12) = 56,116151, count(12) = 5
d(13) = -34,207974, Theta(13) = 66,658385, count(13) = 4
d(14) = -31,793266, Theta(14) = 75,327372, count(14) = 3
d(15) = -29,591818, Theta(15) = 80,537678, count(15) = 1
d(16) = -25,500000, Theta(16) = 90,000000, count(16) = 2
d(17) = -19,329519, Theta(17) = 102,688474, count(17) = 4
d(18) = -24,000000, Theta(18) = 0,000000, count(18) = 1
d(19) = 7,826238, Theta(19) = 116,565051, count(19) = 4
d(20) = -11,407262, Theta(20) = 114,581398, count(20) = 13
d(21) = 12,992109, Theta(21) = 159,881821, count(21) = 2
d(22) = 9,700962, Theta(22) = 154,557102, count(22) = 3
d(23) = -16,559410, Theta(23) = 107,190172, count(23) = 2
d(24) = -8,171513, Theta(24) = 122,270354, count(24) = 19
d(25) = -12,808697, Theta(25) = 99,634119, count(25) = 3
d(26) = -14,000000, Theta(26) = 90,000000, count(26) = 2
d(27) = -14,725725, Theta(27) = 75,009180, count(27) = 2
d(28) = -11,286251, Theta(28) = 106,138863, count(28) = 3
d(29) = -14,975486, Theta(29) = 81,203788, count(29) = 2
d(30) = -14,304688, Theta(30) = 65,816770, count(30) = 2
d(31) = -12,469629, Theta(31) = 51,015857, count(31) = 4
d(32) = -9,796119, Theta(32) = 38,451408, count(32) = 34
d(33) = -7,852002, Theta(33) = 31,227236, count(33) = 4
d(34) = -14,263994, Theta(34) = 60,255119, count(34) = 1
d(35) = -11,667262, Theta(35) = 45,000000, count(35) = 10
d(36) = 12,521981, Theta(36) = 116,565051, count(36) = 1
d(37) = 5,369592, Theta(37) = 45,000000, count(37) = 32
d(38) = 12,197438, Theta(38) = 11,602833, count(38) = 3
d(39) = 13,000000, Theta(39) = 0,000000, count(39) = 3
d(40) = -13,472145, Theta(40) = 169,457111, count(40) = 3
d(41) = 7,738041, Theta(41) = 38,844008, count(41) = 35
d(42) = 9,101753, Theta(42) = 35,860345, count(42) = 9
d(43) = 11,408782, Theta(43) = 19,557102, count(43) = 3
d(44) = -10,000000, Theta(44) = 143,130102, count(44) = 1
d(45) = -9,026346, Theta(45) = 132,605645, count(45) = 15
d(46) = -13,126083, Theta(46) = 158,855017, count(46) = 3

d(47) = -11,663541, Theta(47) = 146,602833, count(47) = 3
d(48) = 5,656854, Theta(48) = 135,000000, count(48) = 1
d(49) = -4,886602, Theta(49) = 120,234759, count(49) = 10

D'abord, il faut remarquer qu'avec cette nouvelle méthode, on arrive à réduire la taille dynamique du tableau de Hough (60 éléments seulement). C'est un signe qu'on arrive à mieux regrouper les éléments.

Les éléments qui pourraient être associés à la première ligne principale à 37° (la ligne la plus haute parmi les trois lignes à 37°) sont :

d(2) = -38,673665, Theta(2) = 38,760131, count(2) = 5
d(4) = -38,486812, Theta(4) = 45,000000, count(4) = 7
d(12) = -36,166221, Theta(12) = 56,116151, count(12) = 5

Les éléments qui pourraient être associés à la deuxième ligne principale à 37° sont :

d(32) = -9,796119, Theta(32) = 38,451408, count(32) = 34
d(33) = -7,852002, Theta(33) = 31,227236, count(33) = 4
d(35) = -11,667262, Theta(35) = 45,000000, count(35) = 10

Les éléments qui pourraient être associés à la dernière ligne principale à 37° sont :

d(41) = 7,738041, Theta(41) = 38,844008, count(41) = 35
d(42) = 9,101753, Theta(42) = 35,860345, count(42) = 9

Les éléments qui pourraient être associés à la première ligne principale à 124° (la ligne la plus à gauche) sont:

d(10) = -6,663309, Theta(10) = 126,764818, count(10) = 122
d(24) = -8,171513, Theta(24) = 122,270354, count(24) = 19
d(45) = -9,026346, Theta(45) = 132,605645, count(45) = 15

Les éléments qui pourraient être associés à la deuxième ligne principale à 124° (la ligne la plus à droite) sont:

d(8) = 9,596799, Theta(8) = 126,983749, count(8) = 138
d(11) = 9,256154, Theta(11) = 122,463228, count(11) = 40

Contrairement au cas précédent, cette nouvelle méthode nous permet de séparer facilement les lignes à 124° en deux groupes. Grâce à la translation d'axe, nous pouvons employer simplement les formules usuelles de test de similarité (différence en d et en θ) pour faire cette séparation sans commettre d'erreur. Pour illustrer ce fait, nous avons pris dans le tableau de Hough tous les éléments (5) qui ont des angles près de 124° et nous avons essayé de présenter tous ces éléments sous forme de droites comme à la figure 3.9.

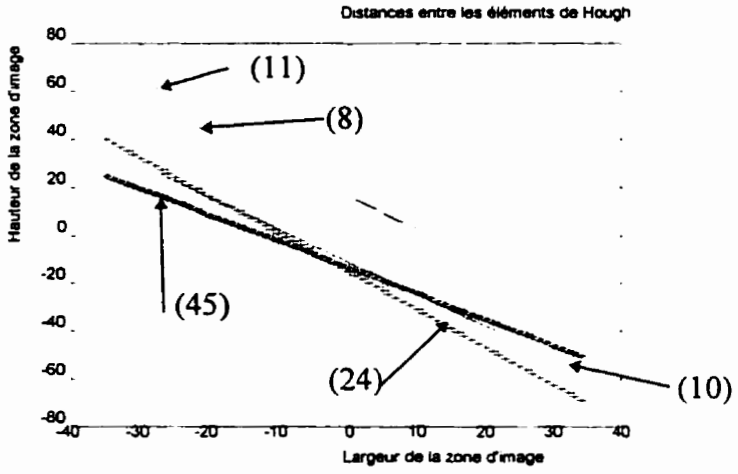


Figure 3.9 Illustration de six éléments typiques du tableau de Hough selon la méthode modifiée

Tel qu'illustré à la figure 3.9, ces cinq éléments appartiennent à l'un des deux groupes et ils sont séparés par un espace très évident. C'est seulement lorsque leurs distances en d et en θ sont inférieures aux seuils Δd et Δθ respectivement que nous pouvons nous permettre de regrouper des éléments de Hough.

Malgré le fait que le test de similarité devienne plus facilement applicable grâce à la translation d'axe, les équations 3.37 et 3.38 contiennent encore un autre facteur qui rend le test de similarité inefficace. C'est le terme de différence en cosinus. Cette différence couvre l'intervalle [0, 2] en valeur absolue. Par conséquent, même si y_B ou y_E (de l'équation 3.37 ou 3.38) étaient de l'ordre de 10, la différence calculée par les

équations 3.37 ou 3.38 pourrait quand même donner une valeur de l'ordre de 0 à 20, ce qui dépassera facilement le seuil Δd. Bref, pour que les formules de test restent réellement efficaces, les droites parallèles à être regroupées devront satisfaire les conditions suivantes :

$$1-\cos(|\theta 1-\theta 2|) < \sin(\Delta\theta) \tag{3,44}$$

$$\sin(|\theta 1-\theta 2|)<\sin(\Delta\theta) \tag{3,45}$$

$$|d1-d2| <\Delta d \tag{3,46}$$

3.6 CONCLUSION

Nous avons vu que la transformation de Hough est analytiquement efficace pour extraire les paramètres caractérisant des lignes droites. Cependant, les méthodes conventionnelles d'implantation de cette transformation génèrent plusieurs sources d'erreurs importantes à savoir l'erreur de quantification et l'erreur d'arrondissement des éléments de Hough. En analysant l'efficacité du test de similarité, nous avons développé une statégie qui consiste à faire une translation d'axe. Les résultats d'exécution sur une image synthétique ont démontré que notre façon d'implanter minimisait la quantité de mémoire requise en plus d'être plus précise dans l'extraction des paramètres.

Cependant, on peut remarquer que, même avec la nouvelle méthode, il existe encore des éléments de Hough qui sont redondants et, par conséquent, qui mériteraient d'être regroupés davantage. Prenons, par exemple, le cas de la première droite à 124°. Nous avons au total trois éléments de Hough [10, 24, 45] qui représentent cette même droite. Nous verrons au chapitre suivant que de tels phénomènes sont inévitables et nous montrerons ce qu'il faut faire pour remédier à ce problème.

Chapitre 4

Algorithmes et implantations

4.1 OBJECTIFS

Nous venons de voir les deux outils mathématiques fondamentaux qui sont le modèle d'énergie de Morrone et la méthode de Hough modifiée dans le chapitre 2 et 3 respectivement. Nous avons vu en détail la performance intéressante de ces deux outils. Nous montrerons maintenant comment nous avons fait pour mettre en application ces outils théoriques pour détecter les points d'intersection de rues.

Nous avons vu dans le premier chapitre que, pour détecter les points d'intersection de rues, nous devons détecter d'abord la position graphique des quatre coins extrêmes de la carte. Déjà à cette étape, nous pouvons entreprendre un premier alignement global de l'image de la carte, dans le cas éventuel où les positions graphiques de ces quatre coins sont distordues lorsqu'on les compare avec leurs coordonnées

géographiques respectives. La façon simple de faire une telle comparaison est d'utiliser les informations concernant la résolution graphique PPP (Pixel Par Pouce) de l'image, l'échelle de la carte et les quatre coordonnées géographiques des quatre coins de la carte pour estimer les distances théoriques entre les quatre coins en terme de pixels. En comparant ces distances avec celles déduites à partir des quatre coins détectés par notre détecteur, nous pouvons déterminer le taux de distorsion global appliqué à chaque coin de la carte et aligner l'image globalement.

Les informations concernant le PPP, l'échelle de la carte et les coordonnées géographiques des quatre coins de la carte sont les données relatives à chaque carte. Elles sont emmagasinées dans un fichier texte que l'on nomme base de données relatives (BDR). On a aussi besoin d'un fichier texte contenant les coordonnées géographiques des intersections de rues. Ce fichier est nommé base de données générale (BDG). Nos partenaires industriels sont responsables de la validité de ces deux types de données. Pour la fin de la simulation, nous devons créer nous-mêmes ces valeurs. La méthode de génération de ces données de base sera présentée brièvement à la section 4.2.

Une fois que l'image est alignée de façon globale, ses quatre coins extrêmes peuvent être utilisés comme points de référence pour déduire les formules de conversions des coordonnées géographiques à des coordonnées graphiques et vice versa pour tous points à l'intérieur de la carte. La méthode de déduction des formules de conversion sera discutée à la section 4.3.

Puisque nous avons implanté et testé les algorithmes utilisant les outils de la librairie TIFF de xv pour les opérations de lecture et d'écriture des fichiers TIFF, nous devons présenter, tout d'abord, à la section 4.4 la structure de données PICINFO que xv utilise pour emmagasiner les informations relatives à une image donnée. La connaissance de cette structure est importante car toutes les implantations utiliseront cette structure de données.

Nous réservons trois sections (4.5 à 4.7) pour parler des méthodes d'implantations du modèle d'énergie de Morrone. La section 4.8 concerne la nouvelle méthode de construction de l'accumulateur de Hough. Malgré que la nouvelle méthode de construction de l'accumulateur de Hough requiert moins d'espace de mémoire, elle forme un accumulateur de Hough ayant plusieurs éléments redondant qui pourraient être regroupés davantage. Nous verrons à la section 4.9 comment nous pouvons remédier à ce problème inévitable.

Les deux dernières sections 4.10 et 4.11 sont réservées à la discussion de la méthode de détermination des points de croisements entre les éléments de Hough et de la méthode d'heuristique pour ne garder qu'une seule réponse (un seul point) parmi un ensemble de points de croisement. Le chapitre va être terminé par une petite conclusion.

4.2 LES ENTRÉES INITIALES

Nous avons vu dans le chapitre 1 qu'à part du fichier d'image contenant l'image de la carte, il nous faut également deux sources de références que nous appelons BDG et BDR. BDG est une base de données qui contient les coordonnées géographiques des intersections de rue selon la référence géostationnaire. Ainsi, la rue Papineau et la rue Jean-Talon doivent se croiser à une latitude et une longitude précise. BDG contient donc des informations stables qui sont indépendantes de l'image avec laquelle on opère.

Nom de rue 1	*Nom de rue 2*	*Longitude(^{0})*	*Latitude (^{0})*
Jean-Talon	*Papineau*	*xxx.xxxxxxxx*	*yyy.yyyyyyyy*

Par contre, la base BDR ne contient que des données relatives à une carte telles la résolution en PPP (pixel par pouce) , l'échelle S_map (kilomètre par pouce) et les quatre coordonnées géographiques des quatre coins extrêmes de carte.

Pour l'instant, nous n'avons pas accès à BDG, tandis que la base BDR n'existe pas. Pour la fin de simulation nous avons généré deux fichiers texte qui peuvent remplacer temporairement BDG et BDR. Pour la fin de la simulation, nous nous sommes servi de notre habitude de raisonnement logique pour déterminer (mesurer par règle) la variable kilomètre par pixel qui n'est rien d'autre que la relation :

$$\left[\frac{\text{km}}{\text{pixel}}\right] = \frac{\left[\frac{\text{km}}{\text{pouce}}\right]}{\left[\frac{\text{pixel}}{\text{pouce}}\right]} = \frac{\text{S\_map}}{\text{PPP}} \qquad (4.1)$$

Tous les outils de dessins commerciaux nous permettent d'extraire la résolution PPP d'une image (sous le nom de dpi). La figure 4.1 montre la fenêtre *Image Info* de CorelPHOTO-PAINT©1993 qui extrait toutes les informations pertinentes de l'image sp1091nw.tif. Ainsi, la résolution de cette image est 300 PPP.

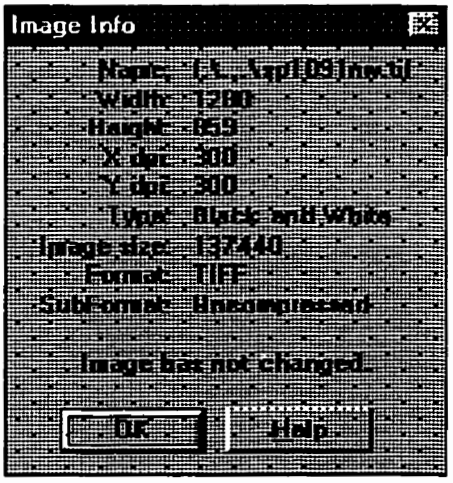


Figure 4.1 ***Mesure de la résolution graphique PPP d'une image***

De l'autre côté, toutes les cartes géographiques devraient avoir leur échelle S_map clairement spécifiée. Une fois la quantité S_map trouvée, il ne reste plus que d'appliquer la formule 4.1 pour déduire la quantité de kilomètre par pixel. Toutefois,

pour la fin de la simulation, cette dernière quantité n'est pas toujours disponible. Afin de contourner ce problème, nous devons évaluer directement la quantité kilomètre par pixel de la carte à analyser en divisant la largeur moyenne de rue (en kilomètre) par la largeur moyenne des rues en pixels.

Cette quantité est indispensable pour estimer les distances équivalentes en kilomètre entre les coins de la carte. Pour évaluer ces distances, nous supposons, de façon arbitraire, que la position graphique d'un des coins (C0 par exemple) soit clairement identifiée et que ce coin correspond à des coordonnées géographiques précises (C0_lon,C0_lat). Nous mesurons ensuite la distance en terme de pixels entre C0 et un des coins adjacents (C1 par exemple) par des outils de dessins commerciaux comme CorelPHOTO-PAINT©1993. La distance en kilomètre entre les deux coins C1 et C0 (C1C0km) n'est rien d'autre que le produit entre S_map et la distance entre C1 et C0 en pixels. La distance C2C1km peut être déterminée de la même manière.

Ensuite, nous choisissons une valeur quelconque pour la latitude du coin C1 (C1_lat) pourvu que ce choix respecte la contrainte suivante :

$$C1\_lat \leq C0\_lat + \frac{C1C0km}{R_T} \tag{4.2}$$

où R_T est le rayon de la terre

Une fois C1_lat définie, on peut utiliser le théorème de Pythagore pour déterminer la longitude du coin en haut à gauche (C1_lon) sous la contrainte de C1C0km selon la relation :

$$C1\_lon = C0\_lon + \frac{180}{\pi}\sqrt{\left(\frac{C1C0km}{R_T}\right)^2 - \left(\frac{\pi}{180}(C1\_lat - C0\_lat)\right)^2} \tag{4.3}$$

Cette équation vient de fixer le degré de longitude de C1 plus à l'est que celui du coin C0. Comme on peut remarquer, ce choix est très arbitraire. Pour générer les coordonnées géographiques des deux autres coins de la carte, il faut supposer que C1C0km=C2C3km et C2C1km=C3C0km. En suite, nous calculons l'angle de l'inclinaison β de l'axe C1C0 par rapport à l'axe vertical par la relation :

$$\beta = \tan^{-1}\left(\frac{C0\_lon - C1\_lon}{C0\_lat - C1\_lat}\right) \tag{4.4}$$

Étant donné que la carte est rectangulaire, l'inclinaison β de l'axe C1C0 est aussi l'inclinaison de toute la carte. Connaissant cette propriété, nous pouvons évaluer les coordonnées géographiques du coin C2 (C2_lon,C2_lat) par les formules suivantes :

$$C2\_lat = C1\_lat - \frac{C2C1km \cdot \sin(\beta)}{R_T}\frac{180}{\pi} \tag{4.5}$$

$$C2\_lon = C1\_lon + \frac{C2C1km \cdot \cos(\beta)}{R_T}\frac{180}{\pi} \tag{4.6}$$

Ce faisant nous avons délibérément choisi l'orientation de la carte penchée vers le bas comme illustrée à la figure 4.2.

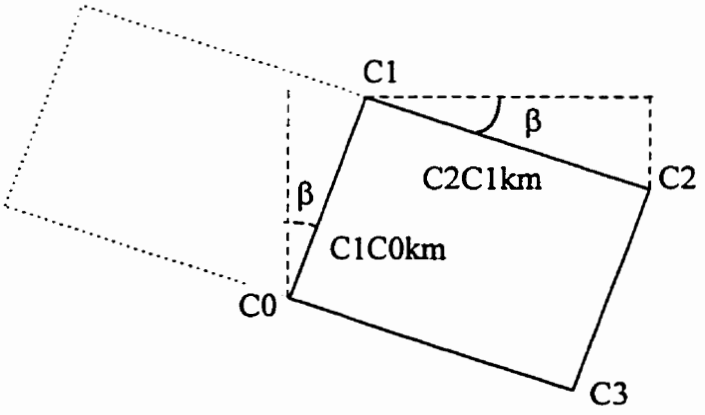


Figure 4.2 Génération des coordonnées géographiques des 4 coins de la carte pour la simulation

Les coordonnées géographiques du coin en bas à droite (C3_lon, C3_lat) peuvent être déterminées de la même manière par les formules :

$$C3\_lat = C0\_lat - \frac{C2C1km \cdot \sin(\beta)}{R_T} \frac{180}{\pi} \tag{4.7}$$

$$C3\_lon = C0\_lon + \frac{C2C1km \cdot \cos(\beta)}{R_T} \frac{180}{\pi} \tag{4.8}$$

Ce qui est important de noter ici, c'est qu'en appliquant ces formules, on a délibérément choisi l'orientation de la carte penchée vers le bas à droite, malgré qu'il existe plusieurs combinaisons possibles de (x,y) qui respectent les contraintes de C1C0km, C2C1km et C2C0km. La fenêtre rectangulaire grise de la figure 4.2 représente une autre orientation possible.

Une fois que les coordonnées géographiques des quatre coins extrêmes sont déterminées, nous les enregistrons dans le fichier texte « bdr » qui respecte le format suivant :

#Commentaire : Cn | degre Nord | degre Est
C0 -17 179
C1 -16,99 179,0155326
C2 -16,9766305 179,0069253
C3 -16,9866305 178,9913927

Pour être plus cohérent, ce fichier doit contenir toutes les informations relatives à la carte à analyser, mais, pour la fin de la simulation, seulement les quatre coordonnées géographiques sont utilisées pour déduire les formules de conversion (u,v) ←→ (i,j).

Comme nous avons supposé précédemment que C2C1km est égale à C3C0km et C1C0km est égale à C3C2km, ceci implique que la carte idéale devrait avoir comme largeur C2C1km et comme longueur C1C0km. Ce choix vient imposer les positions

graphiques idéales (sans distorsion) des quatre coins extrême de la carte. De ce fait, si nous considérons que la position graphique du coin C0 est juste, alors nous pourrons écrire :

$$C1x=C0x\ ;C1y=C0y+C1C0pic \tag{4.9}$$

$$C2x=C1x+C2C1pic;C2y=C1y \tag{4.10}$$

$$C3x=C2x\ ;C3y=C0y \tag{4.11}$$

Nous avons, jusqu'à maintenant, déterminé les quatre coordonnées géographiques des quatre coins extrêmes de la carte pour déduire éventuellement les formules de conversion. Nous venons d'évaluer également les positions graphiques idéales des quatre coins extrêmes de la carte. Si la carte à analyser n'est pas distordue globalement, alors les coordonnées graphiques des quatre coins extrêmes de la carte, mesurées par règle (ou par souris) sur la carte réelle, devraient correspondre exactement aux coordonnées graphiques idéales, calculées à l'aide des équations 4.9 à 4.11. Peu importe le niveau de distorsion, nous pouvons toujours supposer que notre carte n'est pas idéale de telle sorte qu'elle nécessite forcément un premier alignement global utilisant les positions graphiques des quatre coins extrêmes mesurées sur la carte réelle comme points source et les quatre coordonnées graphiques idéales comme points destinataires. Il ne reste plus qu'à sauvegarder ces informations (source, destination) pour chaque coin de carte dans le fichier type texte nommé « warppnt.dat » selon le format.

Source$_x$ ***(en terme de pixel)***	***Source$_y$*** ***(en terme de pixel)***	***Destination$_x$*** ***(en terme de pixel)***	***Destination$_y$*** ***(en terme de pixel)***
mmm	nnn	iii	jjj

Le fichier « warppnt.dat » contient donc des points de contrôle qui vont être utilisés par l'algorithme d'alignement développé par l'équipe du CRIM pour corriger

globalement les distorsions. D'ailleurs, ce même fichier va être utilisé pour emmagasiner d'autres points de contrôle plus tard pour des futurs alignements locaux.

C'est une des façons de trouver rapidement les coins de destination. Nous nous contentons de ces résultats, car il s'agit là d'un artifice pour notre recherche. La partie principale du travail va être présentée au détail à partir de la section suivante.

4.3 FORMULES DE CONVERSION ENTRE COORDONNÉES GÉOGRAPHIQUES ET COORDONNÉES GRAPHIQUES

4.3.1 Nécessité

Nous avons dit dans les chapitres précédents que les intersections de rues constituent pour nous les points de contrôle à être utilisés par l'algorithme d'alignement. Cela étant dit, qu'il est impensable de balayer toute l'image pour détecter toutes les intersections de rues possibles. En effet, il est plus efficace de sélectionner d'abord dans le fichier BDG les coordonnées géographiques des intersections de rues qui sont susceptibles d'être présentes dans l'image à analyser. Pour vérifier si cette intersection existe dans l'image, il faut que nous soyons capables de convertir ces coordonnées géographiques à des coordonnées graphiques en se servant des quatre points de références discutés plus haut. Ainsi, lorsqu'une intersection de rue existe dans l'image à analyser, la position graphique équivalente de cette intersection devrait pointer à une région qui contient un tel événement. Les formules de conversions sont donc très importantes pour vérifier si une intersection de rue quelconque se trouve dans l'image analysée. De plus, les formules de conversion sont essentielles pour nous guider à commencer notre détection dans la région la plus probable. Désormais, nous allons parler de deux formules de conversion continuellement pour désigner une seule

conversion, parce que chaque conversion doit être faite sur deux composantes (u et v) ou (i et j). Le résultat de la conversion donne une position graphique qui devient le centre d'une fenêtre d'analyse de 70x70 pixels que nous utiliserons pour détecter les coordonnées graphiques actuelles de l'intersection de rue en question. La taille de la fenêtre d'analyse est fixée à 70x70 pixels, car, en moyenne, nous remarquons qu'il existe une et une seule intersection de rues en dedans de cette dimension. Les formules de conversions sont basées sur la méthode d'interpolation linéaire dite de triangulation (Goshtasby, 1987). Selon une étude récente de Detlef R. et Heinrich M. (1995), la méthode de triangulation est la méthode la plus performante en temps de calcul et en précision. Cette méthode consiste à former des groupes de trois points à partir d'un ensemble de points de contrôle. Un point de contrôle (x,y) peut être exprimé par deux façons : la première est en coordonnées graphiques (i,j), et la deuxième est en coordonnées géographiques (u,v). Selon le sens de conversion, (u,v) → (i,j) ou (i,j) → (u,v), on va considérer (u,v) comme point de départ et (i,j) comme point d'arrivée et vice versa. Il existe des critères structurés pour regrouper les points en groupe de trois de façon optimale. Chaque groupe de trois points forme une zone triangulaire et détermine les deux formules de conversion applicables à tous points à l'intérieur de cette zone. Nous allons ici résumer les étapes conduisant à ces formules. Supposons que l'on veut déduire les deux formules de conversion des coordonnées géographique à des coordonnées graphiques pour tous les points à l'intérieur de la zone triangulaire définie par trois vertex (u_1,v_1), (u_2,v_2), (u_3,v_3) ayant les trois points cibles (i_1,j_1), (i_1,j_2), (i_3,j_3). La solution revient à chercher les composants normaux d'un plan qui passe par (u_1,v_1,i_1), (u_2,v_2,i_2), (u_3,v_3,i_3) et (u_1,v_1,j_1), (u_2,v_2,j_2), (u_3,v_3,j_3) respectivement. Chaque plan est défini selon

$$A_i u + B_i v + C_i i + D_i = 0 \text{ et } A_j u + B_j v + C_j j + D_j = 0 \qquad (4.12)$$

$$\Longrightarrow i = \frac{-D_i - A_i u - B_i v}{C_i} \quad \text{et} \quad j = \frac{-D_j - A_j u - B_j v}{C_j} \qquad (4.13)$$

Suite à quelques simplifications, nous arrivons à des formules qui déterminent A, B, C et D comme suivent :

$$A_i = \begin{vmatrix} v_1 & i_1 & 1 \\ v_2 & i_2 & 1 \\ v_3 & i_3 & 1 \end{vmatrix} \quad \text{et} \quad A_j = \begin{vmatrix} v_1 & j_1 & 1 \\ v_2 & j_2 & 1 \\ v_3 & j_3 & 1 \end{vmatrix} \tag{4.14}$$

$$B_i = -\begin{vmatrix} u_1 & i_1 & 1 \\ u_2 & i_2 & 1 \\ u_3 & i_3 & 1 \end{vmatrix} \quad \text{et} \quad B_j = -\begin{vmatrix} u_1 & j_1 & 1 \\ u_2 & j_2 & 1 \\ u_3 & j_3 & 1 \end{vmatrix} \tag{4.15}$$

$$C_i = \begin{vmatrix} u_1 & v_1 & 1 \\ u_2 & v_2 & 1 \\ u_3 & v_3 & 1 \end{vmatrix} \quad == \quad C_j = \begin{vmatrix} u_1 & v_1 & 1 \\ u_2 & v_2 & 1 \\ u_3 & v_3 & 1 \end{vmatrix} \tag{4.16}$$

$$D_i = -\begin{vmatrix} u_1 & v_1 & i_1 \\ u_2 & v_2 & i_2 \\ u_3 & v_3 & i_3 \end{vmatrix} \quad \text{et} \quad D_j = -\begin{vmatrix} u_1 & v_1 & j_1 \\ u_2 & v_2 & j_2 \\ u_3 & v_3 & j_3 \end{vmatrix} \tag{4.17}$$

Dans notre application, ce sont les quatre coins extrêmes de carte qui jouent le rôle de points de vertex car ils sont les points les plus extrêmes. Puisqu'une carte est supposée être rectangulaire, la disposition des coins extrêmes de carte divise de manière évidente une carte en deux zones triangulaires idéales. La figure 4.3 rappelle la disposition des coins et illustre les deux zones de triangulation.

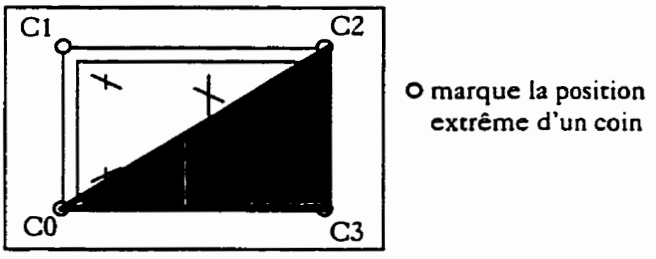


Figure 4.3 ***Convention des positions des coins extrêmes d'une carte et deux façons de regrouper les coins.***

4.3.2 Terminologies utilisés dans l'algorithme de conversion (u, v) <-> (i, j)

Avant de présenter l'algorithme de déduction des formules de conversion, nous profitons cette section pour introduire les terminologies utilisés. Dans notre programme, nous choisissons de grouper les quatre coins selon [C0 C2 C3] et [C0 C1 C2] (figure 4.3). Chaque zone va former des ensembles $\{A_{i/j}, B_{i/j}, C_{i/j}, D_{i/j}\}$ différents de sorte que les points dans la zone plus basse ne peuvent être convertis par les formules découlant des trois points de contrôle plus haut. C'est la raison pour laquelle, nous devons séparer le cas de la zone haute (up) et le cas de la zone basse (dn). Voici la convention des termes utilisés pour les différents cas de conversion :

Axup, Bxup, Cup, Dxup : pour calculer le composant i dans une conversion (u,v) -> (i,j). Ces composants sont applicables aux points de source situés dans la zone supérieure seulement.

Ayup, Byup, Cup, Dyup : pour calculer le composant j dans une conversion (u,v) -> (i,j). Ces composants sont applicables aux points de source situés dans la zone supérieure seulement.

axup, bxup, cup, dxup : pour calculer le composant u dans une conversion (i,j) -> (u,v). Ces composants sont applicables aux points de source situés dans la zone supérieure seulement.

ayup, byup, cup, dyup : pour calculer le composant v dans une conversion (i,j) -> (u,v). Ces composants sont applicables aux points de source situés dans la zone supérieure seulement.

Axdn, Bxdn, Cdn, Dxdn : pour calculer le composant i dans une conversion (u,v) -> (i,j). Ces composants sont applicables aux points de source situés dans la zone inférieure seulement.

Ayup, Byup, Cup, Dyup : pour calculer le composant j dans une conversion (u,v) -> (i,j). Ces composants sont applicables aux points de source situés dans la zone inférieure seulement.

axup, bxup, cup, dxup : pour calculer le composant u dans une conversion (i,j) -> (u,v). Ces composants sont applicables aux points de source situés dans la zone inférieure seulement.

ayup, byup, cup, dyup : pour calculer le composant v dans une conversion (i,j) -> (u,v). Ces composants sont applicables aux points de source situés dans la zone inférieure seulement.

Désormais, à cause de la réciprocité des deux cas de conversion (u,v) ←→ (i,j), nous continuerons notre discussion sur un seul type de conversion (u,v) → (i,j). Les formules inverses peuvent être facilement déduites.

Étant donné un point géographique, il faut déterminer s'il se situe dans la zone supérieure ou inférieure pour pouvoir appliquer les bonnes formules. La règle générale consiste à trouver (m,b) de l'équation y=mx+b qui passe par C0 et C2. Ainsi, on peut tester aisément si un point se trouve dans la zone supérieure ou non.

4.3.3 Les algorithmes pour les deux formules de conversion (u,v) -> (i,j) et les initialisations

Pour assurer la clarté et la simplicité, nous allons présenter l'algorithme utilisant les paramètres A, B, C, D sans indices. L'algorithme s'applique aussi bien pour les paramètres de la zone supérieure que pour les paramètres de la zone inférieure, pour la composante en x que pour la composante en y, pour les paramètres en majuscule que pour les paramètres en minuscule.

4.3.3.1 Algorithme de conversion des coordonnées géographiques à des coordonnées graphiques

```
Si ABCD ne sont pas définis ALORS
        aller lire des coordonnées géographiques (getdbinfo) dans le fichier «bdr »
        faire confirmer par un affichage par windows (Code du CRIM)
        calculer tous les {ABCD} selon les équations de 4.14 à 4.17
        Si la lecture n'est pas réussie ALORS
                retourner ECHEC
        fin si
fin si
/*ABCD sont définis*/
Si les coordonnées géographiques à l'entrée correspondent à la zone triangulaire supérieure
ALORS
        Calculer i, j selon 4.13 utilisant A,B,C,D de la zone supérieure
Sinon calculer i, j selon 4.13 utilisant A,B,C,D de la zone inférieure
retourner SUCCES
```

L'interface de ce sous-programme est :

```
int uv2ij(double *u, double *v)
```

Les coordonnées sont passées par paramètres. Les {ABCD} sont à calculer une fois seulement. Ils doivent être recalculés uniquement lorsqu'on change l'image de la

carte auquel cas il faut refaire la détection des quatre coins extrêmes de la carte. Le fichier « bdr» doit être également remis à jour pour correspondre à la nouvelle carte. Pour l'instant, nous forçons les opérateurs à préparer le fichier de BDR et le sauvegardent sous le nom de «bdr». Éventuellement, pour atteindre un niveau d'automatisme plus élevé, les opérateurs peuvent préparer le fichier de BDR et le sauvegardent sous le nom qu'ils désirent. Cependant, ils doivent spécifier le chemin complet pour accéder à ce fichier dans un champ particulier du format TIFF.

4.3.3.2 Procédures d'initialisation

Nous pouvons remarquer que lorsqu'on ouvre un nouveau fichier de carte, les valeurs ABCD et les positions des quatre coins extrêmes de la carte ainsi que les quatre coordonnées géographiques correspondant aux quatre coins extrêmes de la carte, que nous avons actuellement dans la mémoire, nécessitent d'être réexaminées, car ces donnés peuvent ne plus correspondre à la nouvelle carte. Il est donc nécessaire de réinitialiser les ABCD et les ref_point (définis par l'interface développée au CRIM) à 0. La structure de ref_point contient entre autres les champs d'informations suivantes :

```
struct Ref_CtrlPtr
{
        ...
        x ;     //coordonnée graphique en colonne
        y ;     //coordonnée graphique en rangée
        lon ;   //coordonnée géographique en longitude
        lat ;   //coordonnée géographique en latitude
        ...
}
```

Selon la convention que nous avons fixé ensemble, la variable ref_point est le nom d'un tableau de 4 éléments tels que le premier élément (ou élément ayant l'indice 0) correspond au coin C0 de la carte et ainsi de suite.

Nous avons intérêt à calculer les ABCD et remplir le tableau de ref_point aussitôt qu'on ouvre une nouvelle image, car il est intéressent d'afficher en permanence les coordonnées géographiques et les coordonnées graphiques de tous les points de l'image. Pour cela, il faut détecter préalablement les positions des quatre coins extrêmes de carte.

Lorsqu'on n'arrive pas à détecter les quatre coins à cause d'un degré de distorsion trop élevé, on peut s'assurer que le programme développé par l'équipe du CRIM fasse des actions (procédures) nécessaires pour obliger les usagers à vérifier et à effectuer la dernière modification ou remplissage des informations reliées aux quatre coins extrêmes de carte par le biais d'une fenêtre Question/Réponse.

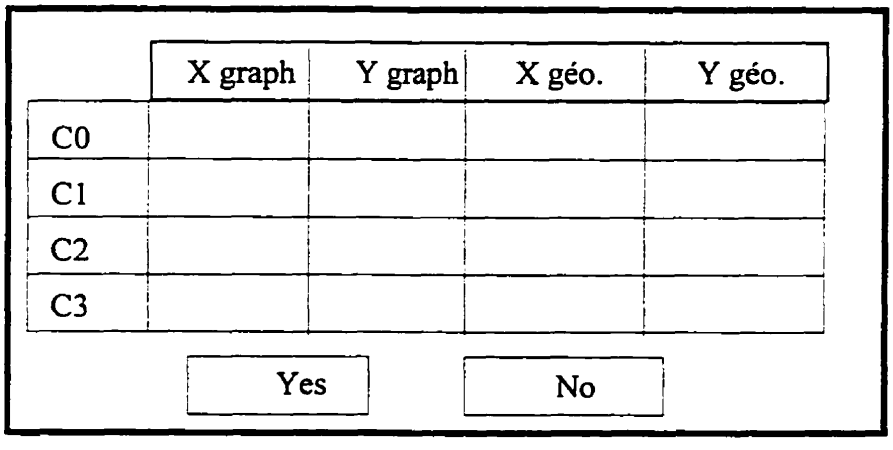


Figure 4.4 ***L'interface développée par l'équipe du CRIM pour la lecture et les modifications des informations reliées aux 4 coins extrêmes de la carte.***

La réponse « No » (ou refuse) à cette fenêtre va être propagée vers les procédures appelant de telle sorte qu'aucune conversion ne peut être effectuée. La réponse « Yes » signifie que l'usager admet que les quatre groupes d'informations fournies dans le tableau sont exacts, auquel cas nous pouvons calculer les valeurs de ABCD, si nécessaire, pour déduire le résultat de la transformation..

4.4 LA REPRÉSENTATION DES PIXELS DANS XV© 1993

L'interface que l'équipe du CRIM a utilisée pour exécuter les programmes d'application d'images est le xv© 1993 de Unix. Pour simplifier la tâche, nous travaillons également avec cette interface et utilisons la même structure requise par xv, notamment la structure PICINFO.

```
typedef struct {
        BYTE *pic ;             /* image data */
        intxv w, h ;            /* pic size */
        int  type ;             /* PIC8 or PIC24 */
        BYTE r[256],g[256],b[256] ;  /* colormap, if PIC8 */
        intxv normw, normh ;   /* 'normal size' of image file  (normally eq. w,h, except when
doing 'quick' load for icons */
        int  frmType ;          /* def. Format type to save in */
        int  colType ;          /* def. Color type to save in */
        char fullInfo[128] ;    /* Format : field in info box */
        char shrtInfo[128] ;    /* short format info */
        char *comment ;         /* comment text */
        int  numpages ;         /* # of page files, if >1 */
        char pagebname[64] ;    /* basename of page files */
} PICINFO ;
```

La variable pic, pointeur du type BYTE, est le pointeur à un tableau 1D tel que chaque élément du tableau contient l'indice de couleur que cet élément (pixel) a sur l'image. La position en 2D de cet élément est :

$$\text{rangée} = j = \frac{\text{Offset}}{\text{Largeur de l'image}} \tag{4.18}$$

$$\text{colonne} = i = \text{Offset} - i \cdot \text{Offset} \tag{4.19}$$

où Offset est l'indice de l'élément dans le tableau pointé par pic et la division est sans reste (division des nombres entiers), ce qui entraîne des arrondissements automatiques.

Selon la convention de presque tous les éditeurs graphiques, l'origine de l'image (0,0) est toujours le point en haut à gauche et l'axe vertical de l'image est positif vers le bas. De ce fait, le premier élément du tableau pointé par la variable pic (Offset == indice == 0) correspond au premier pixel en haut à gauche de l'image.

La valeur contenue dans la partie de mémoire pointé par (pic+Offset) est considérée comme l'indice des trois tableaux de références :

BYTE r[256], g[256], b[256]

En effet, la vraie couleur qu'un élément a sur l'image est le résultat de la contribution des trois couleurs fondamentales correspondant à l'indice *(pic+Offset) du tableau de référence de trois couleurs fondamentales. L'interface de xv fait tout ce qui est nécessaire pour remplir PICINFO. Il ne nous reste plus qu'à servir des informations emmagasinées dans PICINFO pour répondre spécifiquement au besoin de notre application.

Avec xv, lorsque r[n]=255, b[n]=255, g[n]=255, on a la couleur blanche pour l'indice n, tandis que le triplet (0,0,0) correspond à la couleur noire. Les autres couleurs intermédiaires sont formées par des combinaisons de ces trois couleurs avec des valeurs entre [0..255]. On a déjà vu dans le chapitre 2 que notre algorithme de détection de contours exige que les pixels d'événements aient de la valeur MAXENER (i.e 15) et que les pixels de l'arrière scène aient de la valeur 0. Cette convention est contraire à la convention utilisée par xv. Avec xv, puisqu'un pixel d'événement (couleur noire sur l'image) doit être représenté par le triplet (0,0,0), tandis qu'un pixel de fond (couleur blanche sur l'image) doit être représenté par le triplet (255,255,255), nous devons faire

une étape d'inversion avant appliquer les filtres de Morrone. Cette méthode est simplement décrite par l'équation suivante :

$$\text{nouvelle valeur de pixel} = \frac{255 - \frac{r[] + b[] + g[]}{3}}{17} = 15 - \frac{r[] + b[] + g[]}{51} \qquad (4.20)$$

Ce faisant, on s'assure que notre signal d'entrée va avoir des amplitudes se situant dans l'intervalle [0..15]. Dans le cas d'une image en noir et blanc, le pixel d'événement va avoir la valeur 15 et le pixel de l'arrière scène va avoir la valeur 0.

Pour l'instant, nous avons supposé que notre application s'adresse à des images en noir et blanc seulement, c'est-à-dire qu'il est suffisant de faire de la détection de contours à partir d'un des trois éléments du triplet (r,g,b). La façon la plus simple de tenir compte de la contribution de chaque composant r,g,b est évidemment de faire leur somme que l'on divise ensuite par 3. Cette équation va devenir plus ou moins valide lorsqu'on travaille avec des images en couleur, puisque, dans ce cas, il faut effectuer la détection d'arêtes pour chaque composant r,g,b séparément.

Dans l'ensemble de cartes que nous avons à traiter, il existe des cartes qui ont un fond noir et des pixels d'événement blanc. Dans ce cas, nous n'avons pas besoin de faire de l'inversion telle que suggérée par l'équation 4.20, car les niveaux de gris des pixels satisfont déjà l'exigence du modèle d'énergie de Morrone. Pour tester si l'image à analyser est une image à fond noir ou blanc, nous avons développé une petite procédure dont l'interface est :

```
int TestBack(PICINFO In)
```

Cette procédure lit une image TIFF et évalue le nombre de pixels noirs qui apparaissent dans une petite région. Si ce nombre occupe plus que la moitié de la surface de la région, nous pourrons conclure que l'image à analyser a un fond noir ; sinon on conclut que l'image a un fond blanc.

4.5 ALGORITHME DE CONVOLUTION UTILISANT LES FILTRES DE MORRONE DANS LE DOMAINE 2D

Nous venons de voir que notre signal d'entrée est exprimé par un vecteur de ligne 1D. Ce tableau est pointé par le pointeur pic (voir figure 4.5).

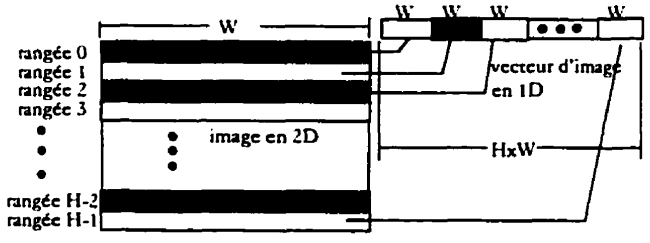


Figure 4.5 Le stockage du contenu de l'image dans un vecteur à une dimension

De l'autre côté, les filtres de Morrones sont difficiles à modéliser par les filtres 2D, surtout lorsque le vecteur du signal d'entrée est déjà exprimé en 1D. Nous avons alors décidé de faire de la convolution entre les filtres pair et impair avec le signal d'entrée dans le sens horizontal d'abord. Ensuite, les mêmes filtres sont appliqués à l'image une deuxième fois, mais dans le sens vertical cette fois. Pour chaque sens, nous évaluons les maximum locaux du signal d'énergie (voir figure 4.6). Le résultat final correspond à l'opération OR entre deux résultats intermédiaires et l'opération ET avec la vérification si ce point d'arête est un pixel d'événement.

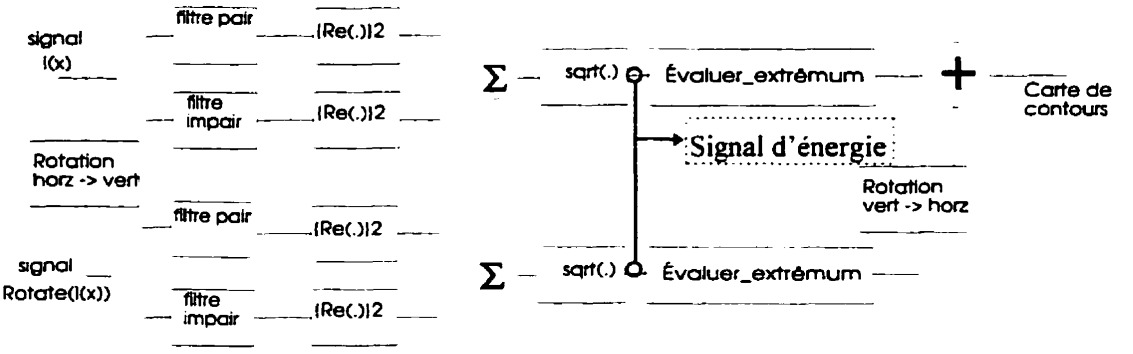


Figure 4.6 Modèle du système de détection d'arêtes

Les filtres Fe (filtre pair) et Fo (impair) sont générés et testés par Matlab (voir chapitre 3). Ils sont sensés être les plus performants. Pour économiser le temps de calcul, nous sommes prêt à gaspiller un peu de mémoire pour emmagasiner préalablement les deux filtres Fe et Fo dans le programme (50 éléments chaque).

Pour faire la convolution, nous avons imaginé que le signal d'entrée est vraiment un signal à 1D, tel qu'illustré à la figure 4.5. Cependant à cause de la propriété d'expansion du domaine de définition du résultat de convolution, nous aurons comme signal résultant un vecteur de longueur WxH+50 et non plus de WxH. Pour remédier à ce problème, nous avons décidé de nous débarrasser les 50 derniers éléments du résultat de convolution pour qu'il n'y reste plus que WxH éléments. Nous ne pouvons faire cette action que lorsque les filtres pair et impair ont des amplitudes nulles vers la fin de leur signal et que les amplitudes non nulles sont beaucoup moins présentes.

Heureusement, nos deux filtres ont des valeurs non nulles pour les 12 premiers éléments seulement. Ils ont des valeurs presque 0 pour les 38 éléments restant. C'est donc un rapport de 75 % des valeurs « non utilisées ». Il suffit d'avoir un vecteur du signal d'entrée ayant au moins 12 derniers éléments égaux à 0 pour obtenir 50 éléments nuls à la fin du signal résultant au quel cas nous aurions la permission de conserver seulement les W+H éléments du signal résultant. Cette condition est généralement très facile à atteindre dans des applications traitant des images numériques.

Il existe des méthodes qui essaient de faire un lien entre la façon d'appeler un pixel dans le domaine 2D et l'accès réel au vecteur 1D. Par exemple, Craig A. Lindley (1991) a conçu une fonction appelée GetPixelFromImage(Inimage, Col, Row), où Inimage est le vecteur du signal d'entrée 1D, Col est le colonne dans lequel se trouve le pixel à rechercher et Row est la rangée dans laquelle se trouve le pixel à rechercher. À ce niveau, nous sommes tous en mesure de concevoir cette dernière fonction. L'auteur de ce livre n'a pas tort d'utiliser une telle fonction pour ses traitements, car les filtres qu'il a utilisé sont surtout les filtres 2D.

4.6 ALGORITHME DE TRANSPOSITION D'IMAGES

Nous venons de voir à la figure 4.6 qu'il nous faut une procédure pour faire la transformation d'image de l'horizontale à la verticale et vice versa. Analysons le problème de la transformation horizontale à verticale d'abord. Il faut se fixer en tête l'idée que les filtres de Morrone balaient toujours l'image de gauche à droite et de haut en bas. Pour effectuer un balayage vertical, il faut renverser l'image comme lorsqu'on transpose une matrice 2D (voir figure 4.7).

$$S=\begin{bmatrix} a & b & c & d & e \\ f & g & h & i & j \\ k & l & m & n & o \end{bmatrix} => \begin{bmatrix} a & f & k \\ b & g & l \\ c & h & m \\ d & i & n \\ e & j & o \end{bmatrix} = D$$

Figure 4.7 La similitude entre la transposée d'une matrice et le renversement d'un vecteur d'image.

On peut observer que l'indice de la rangée et de la colonne de chaque élément de S sont renversés par rapport à D. Par exemple h_S=(2,3) => h_D =(3,2). Même la largeur de S (5) est maintenant devenu la hauteur de D et vice versa. Ainsi, lorsqu'on balaie D dans

le sens horizontal, ceci est équivalent à balayer S dans le sens vertical. L'algorithme de renversement de l'image se résume donc aux quelques lignes suivantes :

Créer un vecteur tampon (T) de longueur WxH
Pour chaque élément dans le vecteur pointé par pic faire
    Déterminer sa rangée i et son colonne j à partir de Offset (voir 4.4)
    Insérer à la position j*W+i de T la valeur de l'élément ayant l'indice Offset dans pic.
        C'est-à-dire T[j*W+i] ← pic[Offset]
Échanger W et H
Réassigner le tout au vecteur pic.

Suite à l'utilisation de cette procédure, nous obtenons un vecteur décrivant l'image renversée. L'avantage de cette méthode est que le système de détection d'arêtes illustré à la figure 4.7 ne devrait faire aucune considération particulière pour traiter ce cas. Le détecteur d'arêtes va considérer le vecteur décrivant l'image renversée comme étant un nouveau vecteur d'image. Évidemment, le prochain appel à cette procédure va ramener le vecteur d'image à sa forme originale.

4.7 INTERPRÉTATION FINALE SUR LE SIGNAL D'ÉNERGIE ET EMMAGASINAGE DES POINTS D'ARÊTES

Pour compléter le module de détection d'arêtes tel que présenté à la figure 4.6, nous devons construire un évaluateur des maximum locaux pour déterminer les endroits où il y a des arrivées en phase (voir chapitre 2) ou encore tout simplement des arêtes. Grâce à performance de nos filtres détecteurs, le signal d'énergie résultant donne une réponse presque évidente quant à l'interprétation.

Malgré cela, la recherche des maximums locaux est depuis toujours une casse-tête pour le monde scientifique, parce qu'il s'agit du problème de détermination du seuil optimal et parce qu'il n'y a rien d'absolu dans ce monde si l'on regarde des choses à une

plus grande échelle. Sachant cela, nous avons défini dès le départ qu'une localité est composée de trois pixels consécutifs.

Par définition, une localité contient un maximum lorsqu'il a un pixel central ayant une valeur d'énergie qui dépasse un certain seuil tout en étant plus élevée que les valeurs d'énergie des deux pixels voisins. Le terme « plus élevé » donne naissance à un autre seuil que nous appelons seuil1 dans le programme. Pour pouvoir détecter le moindre pic, il faut fixer seuil1à une petite valeur (2 unités dans notre cas).

Dans le cas d'une image binaire, le seuil global est plus facile à fixer. Puisque nos filtres sont conçus de telle sorte que le signal d'énergie résultant reste encore dans l'intervalle [0..15]. Un seuil de 7.5 est adéquat pour pouvoir séparer l'image en deux niveaux. Voici le pseudocode de l'algorithme de détection des maximum locaux appliqué à des signaux de 1D :

Soit le signal d'énergie Res et un tableau de tampon V :

```
V[0]← 0, V[finale]← 0
si Res[0]-Res[1] > seuil1 ET Res[0] > seuil ALORS
        V[0]=15
Pour i : 1→ taille de Res -2
        seuil gauche ← Res[i]-Res[i-1]
        seuil droite ← Res[i] - Res[i+1]
        si Res[i] > seuil ET seuil gauche > 0 ET seuil droite >0  ET  (seuil gauche OU seuil
        droite >seuil1) ALORS
                V[i] ← 15
        sinon V[i] ← 0
/* i est l'indice du dernier élément de Res*/
Si Res[i]>seuil ET Res[i]-Res[i-1]> 0 ALORS
        V[i] ← 15 /*équivalent à V[finale] ← 15*/
Retransférer tous les éléments de V dans Res
```

Cet algorithme est utilisé deux fois : la première fois pour détecter les maximums locaux dans le signal d'énergie horizontal et la seconde pour le signal d'énergie dans le

sens vertical. Le résultat final est la combinaison de ces deux résultats intermédiaires. La formule de combinaison est dans notre cas de type:

Res_{Final_i}= pixel d'événement ssi

$Res1_i$OU $Res2_i$ ET $pixel_i$ de PICINFO sont des pixels d'événement. (4.21)

Pour pouvoir observer le résultat de détection d'arêtes, nous devons respecter les conventions de xv et faire des transformations nécessaires sur Res_{Final}. Une fois que nous sommes assurés que notre méthode d'implantation pour la détection de contours fonctionne bien, nous pouvons passer à l'étape de détection de lignes et de coins. Avant cela, il est important de noter (ou mémoriser) les positions des pixels d'arêtes dans un tampon, car ces informations sont indispensables pour les prochaines étapes de détection des coins.

Parce que nous travaillons principalement dans UNIX (car on a ici tous les outils nécessaires pour lire des images de type TIFF), nous n'avons pas eu accès aux librairies «containers » (structure de données généralisée pour les listes liées dynamiquement avec les méthodes de gestion de listes incluses). C'est pourquoi, nous avons choisi la méthode d'implantation la plus simple pour emmagasiner nos résultats intermédiaires. En fait, nous avons employé les tableaux ordinaires (array en C et C++), même si nous sommes conscients que nous devons payer très cher pour le temps de recherche ou de suppression des éléments que nous allons d'ailleurs utiliser un nombre exorbitant de fois. Nous reviendrons sur ce point en temps et lieu dans les prochaines section lorsqu'on parle de l'accumulateur de Hough.

Pour mémoriser la position des pixels d'arêtes, nous avons utilisé un fichier binaire qui a le nom de « edge.dat », car nous ne savons pas a priori le nombre exact des pixels d'arêtes que nous pourrons détecter. Pour minimiser la taille de ce fichier et pour

simplifier la recherche d'éléments dans ce fichier, nous employons la stratégie fondée sur le fait que l'on dispose de deux façons équivalentes pour exprimer la position d'un pixel. En effet, on peut exprimer cette position sous forme de (Rangée, Colonne), mais on peut également exprimer cette position sous forme de (W,Offset) où W est la largeur de l'image et Offset est la combinaison unique de Rangée, Colonne et W d'après les formules 4.18, 4.19 (voir section 4.4) et la formule suivante :

$$\text{Offset} = \text{Rangée} * W + \text{Colonne} \qquad (4.22)$$

Il est donc nettement plus avantageux d'utiliser le dernier type d'expression pour emmagasiner la position des pixels d'arêtes car tous les pixels devraient avoir la même valeur W. Par conséquent, la seule information à être emmagasinée réellement est la valeur Offset de chaque pixel d'arêtes. Quoi qu'il en soit, il faut au moins enregistrer une fois la valeur W pour avoir toutes les données nécessaires pour faire de la conversion inverse.

L'interface de la fonction qui fait tout le travail de détection de contours et de préparation du fichier contenant des pixels d'arêtes est comme suit :

FILE * EdgeListDetect(PICINFO &ImageBuf, uint16 Row, unit16 Col)

Les variables Row et Col désignent la position en haut à gauche de la fenêtre d'analyse (de dimension 70 pixels X 70 pixels). Cette fonction ouvre le fichier « edge.dat » mais ne le ferme jamais. Elle passe ce pointeur « ouvert » aux fonctions qui l'ont appelé. Dans tout le programme, il y a deux fonctions qui font des appels explicites à cette fonction. Ce sont les fonctions « CornerDetect » et « LocateCorner ».

Quand on fait appel à la fonction « EdgeListDetect », on est tous conscients que les anciennes données enregistrées dans « edge.dat » actuellement ne nous intéressent plus, il nous faut les nouvelles. Alors, la seule façon la plus simple pour supprimer les

anciennes données dans « edge.dat » est de le fermer et de le rouvrir. Désormais, la fonction qui a besoin des nouvelles données sur les arêtes doit prendre la responsabilité de fermer le fichier « edge.dat ». Hormis ces petites complications, l'utilisation de ce type de représentation pour emmagasiner les pixels d'arêtes est aussi efficace que les listes chaînées.

On vient de voir la façon de détecter et d'emmagasiner les positions des pixels d'arêtes. Il nous reste maintenant à approfondir les techniques utilisées pour arriver jusqu'à la détermination de la position des coins de rue, s'ils existent.

4.8 CONSTRUCTION DE L'ACCUMULATEUR DE HOUGH

4.8.1 Nécessité

Parmi les points enregistrés dans « edge.dat », on est certain qu'il existe des points d'intersections (entre les segments). Il faut alors trouver des lignes qui passent par ces points et déterminer ensuite les points d'intersection entre ces droites. Pour localiser et quantifier les lignes, on utilise la transformée de Hough.

Comme on a vu à la section 3.3, la transformée de Hough doit être utilisée dans une petite région de telle sorte que les points à analyser ne soient pas trop loin de l'origine virtuelle. L'origine virtuelle est le nom que l'on donne au point central de la fenêtre d'analyse. Pour une analyse donnée, on considère cette origine comme étant (0,0). Tous les autres points à l'intérieur de la fenêtre d'analyse sont par conséquent près de l'origine virtuelle et les paramètres (d, Theta) sont exprimés par rapport à cette référence. La figure 4.8 montre graphiquement l'emplacement de l'origine virtuelle. Cette origine dépend du résultat de la conversion (u,v) → (i,j) d'une intersection de rues.

Le Row_Offset et le Col_Offset représentent la distance en rangée et en colonne entre une origine virtuelle et la vraie origine.

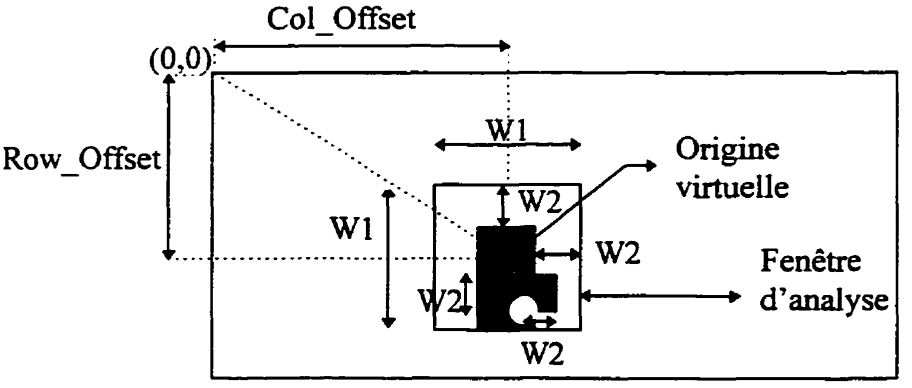


Figure 4.8 Contexte d'analyse

Dans la figure 4.8, la zone grise pâle représente une portion de l'image ayant le coin en haut à gauche comme l'origine (0,0). Il est important de se rappeler que l'axe vertical est positif vers le bas. On se réfère à cet axe par un des noms suivants : y, lat, row, v ,j (voir section 1.2).

Le petit carré blanc représente la grandeur typique d'une fenêtre d'analyse où W1=70 dans notre application. Le centre de cette fenêtre devrait correspondre exactement à l'intersection de rues que l'on examine. Malheureusement, à cause des distorsions, ce centre va être situé plus ou moins près de l'intersection recherchée. Comme on peut voir sur le dessin, la fenêtre d'analyse est un carré tel que chaque côté a une longueur de W1 pixels. On suppose qu'à l'intérieur de la fenêtre d'analyse, il n'existe qu'une et une seule intersection de rue. Pour satisfaire cette exigence, la valeur de W1 doit être choisie de façon appropriée.

La valeur W2 définit la hauteur de la petite fenêtre rectangulaire (noire ayant un trou de demi cercle) qui occupe la partie supérieure du pixel central d'analyse. Tous les

pixels d'arêtes à l'intérieur du rectangle gris foncé vont être considérés consécutivement comme le pixel central d'analyse et examinés de près pour voir avec quel élément de Hough ils pourraient être associés.

Pour déterminer cette association , sans connaître a priori l'orientation des lignes droites ni leur longueur ni le nombre de lignes qui existent dans la fenêtre d'analyse, nous avons opté pour la méthode d'énumération complète. Cette méthode consiste à considérer comme pixel central (un pixel à la fois) tous les pixels d'arêtes à l'intérieur de la fenêtre grise en appelant la fonction « PickEdgeList » dont l'interface est la suivante :

uint32 PickEdgeList(uint32 Offset1, uint32 Offset2, FILE* EdgeFile)

où

Offset1 correspond à la position de début de la rangée qui contient les pixels centraux,

Offset2 correspond à la position de fin de la rangée qui contient les pixels centraux,

EdgeFile est le pointeur vers le fichier edge.dat

Lorsque la fonction « PickEdgeList » ne trouve plus aucun pixel d'arête sur la rangée spécifiée, elle retournera 0 à la fonction appelant. À ce moment, nous devons continuer notre analyse à la rangée suivante, jusqu'à ce qu'on ait balayé toute la surface du rectangle gris foncé.

Pour chaque pixel central donné, nous cherchons et emmagasinons dans un tableau séquentiel tous les pixels voisins qui se situent à l'intérieur de la région rectangulaire noire, qui sont séparés d'une distance PCLOSE (égale à trois pixels dans

notre application) par rapport au pixel central en cours, en utilisant la fonction FindEdgeNeighbor dont l'interface est comme suit :

void FindEdgeNeighbor(uint32* Edges, uint32 Offset1, intxv WPic, FILE* EdgeFile)

où

Edges se réfère à un tableau de type uint32 et de taille fixée à W1,

Offset1 représente la position du pixel central,

Wpic représente la largeur de l'image en terme de pixels,

EdgeFile est le pointeur du fichier « edge.dat ».

Cette fonction retourne un tableau qui contient les positions des pixels voisins du pixel central. Pour chaque pixel voisin, nous devons calculer l'orientation θ de la droite passant par lui et le pixel central.

$$\theta = \tan^{-1}\left(\frac{y_1 - y_2}{x_2 - x_1}\right) \tag{4.23}$$

Il y a une inversion de signes dans cette formule pour tenir compte de l'effet de renversement du système d'axe que nous avons mentionné un peu plus haut. La fonction atan2 incluse dans la librairie de tous les compilateurs C et C++ nous permet de calculer même les angles qui sont près ou égales à ±90°. La fonction atan2 nous retourne le résultat en radian entre [-π,π].

Lorsque nous travaillons avec les fonctions trigonométriques, il faut faire attention aux signes, ce qui alourdit inutilement nos algorithmes. Pour éviter cela, nous

exprimons tous les angles en valeurs positives. En effet, il suffit de remarquer que la version équivalente d'un angle négatif est la valeur de cet angle plus 180°.

C'est seulement après cette normalisation que nous pouvons déterminer la valeur d par la formule (voir chapitre 3)

$$d = x \bullet \sin(\theta) + y \bullet \cos(\theta) \qquad (4.24)$$

A partir de cette étape, toutes les données telles que rangée, colonne, d, x et y doivent être exprimées par rapport à l'origine virtuelle pour minimiser les erreurs de regroupement. Seulement le résultat final va être considéré par rapport à la vraie origine.

Lorsque que le couple (d,θ) est déterminé, nous arrivons à l'étape de mise à jour de l'accumulateur en vérifiant si l'accumulateur de Hough contient un des éléments qui est presque identique au couple (d,θ) actuel. Dans l'affirmative, nous devons mettre à jour l'élément trouvé. Dans le cas où l'accumulateur de Hough ne contient aucun élément qui est presque identique au couple (d,θ) actuel, on doit insérer ce nouveau couple à la queue de l'accumulateur de Hough.

Le terme « presque identique » requiert un peu d'effort pour être correctement modélisé, car on n'est pas en train de calculer la similitude entre une seule quantité d'une variable, mais bien de deux quantités. Heureusement, la translation d'axe (discutée à la section 3.3 et 3.4) nous a permis de dire que deux couples (d,θ1) et (d,θ2) sont presque identiques si et seulement si la distance entre les d et les θ n'excède pas un certain seuil. Pour la distance en d, le seuil est de del_d (trois pixels). Pour la distance en θ, le seuil est de del_Theta/2 ou encore 5°. Les seuils que nous avons choisi sont généralement basés sur l'intuition et l'expérimentation.

Selon notre définition, deux couples (d,θ) sont presque identiques lorsque les deux θ sont presque parallèles, tandis que les deux d sont presque égales en valeur absolue. Les deux θ de deux droites presque parallèles doivent vérifier les contraintes :

$$\sin(|\theta_1 - \theta_2|) < \sin\left(\frac{\Delta\theta}{2}\right) \quad \text{et} \quad 1 - \cos(|\theta_1 - \theta_2|) < \sin\left(\frac{\Delta\theta}{2}\right) \tag{4.25}$$

Le test en sinus est plus performant que le test en cosinus pour les droites horizontales. En revanche, le test en cosinus est plus performant que le test en sinus pour les droites verticales. Pour les cas obliques, les deux test sont aussi bons l'un que l'autre.

En effet, pour les droites horizontales, leur angle θ est habituellement près de 180° ou 0°. Sur le plan trigonométrique une grande variation en θ dans cette condition donne toujours une variation minime en sin(θ). Par conséquent, une simple comparaison en sin(θ) uniquement ou en cos(θ) uniquement n'est pas suffisant pour conclure si deux droites sont parallèles.

La différence en d est plus facile à calculer, on n'a qu'à utiliser la formule suivante :

$$\Delta d = |d_1 - d_2| \tag{4.26}$$

Pour assurer une mise à jour la plus optimale, nous devons chercher dans tous les éléments définis dans l'accumulateur de Hough, l'élément ayant la distance la plus courte avec le couple (d,θ) actuel. C'est ce que fait la fonction Exist.

Cette fonction nous retourne l'indice de l'élément de Hough avec lequel il faut effectuer une mise à jour adéquate: soit par insertion, soit par regroupement. Dans le cas où l'indice retourné correspond à un élément ayant le champ compteur égal à 0, nous sommes dans la situation où il faut faire de l'insertion, car l'accumulateur de Hough n'a

jamais enregistré un tel couple. Dans le cas où l'indice retourné correspond à un élément ayant le champ compteur plus grand que 0, nous sommes dans la situation où il faut faire un regroupement car l'accumulateur de Hough a déjà enregistré un couple semblable. Lorsque la fonction Exist retourne un indice dépassant la longueur de l'accumulateur, elle veut signaler au programme appelant que l'accumulateur de Hough est plein et il faut le réorganiser avant de pouvoir continuer à insérer des nouveaux couples.

L'interface de la fonction Exist est :

int Exist(double d, double Theta, THough *H_Acc) ;

où

d et Theta sont les paramètres de la droite nouvellement calculée,

H_Acc se réfère au tableau de Hough tel que chaque élément de Hough a de la structure suivante :

```
struct THough
{
        double d;
        double Theta;
        uint16 cnt;      /*le champ compteur permet de mieux visualiser le résultat de
l'accumulation */
};
```

Cette fonction calcule la différence entre (d,θ) et chaque élément du tableau de Hough en faisant l'appel à la fonction « ifdiff » dont l'interface est comme suit :

double ifdiff(double d,double Theta,THough H_Acc)

La fonction ifdiff retourne une constante qui est égale à 360 lorsque la similitude entre le couple (d,θ) et l'élément courant du tableau de Hough n'est pas suffisante. Dans

le cas contraire, elle retournera une valeur proportionnelle à la distance mesurée selon la formule :

$$\text{valeur retournée} = |d_1 - d_2| * \sin(\theta_1 - \theta_2) * \frac{180}{\pi} \qquad (4.27)$$

Le facteur 180/pi est un facteur de grossissement sur la valeur retournée pour élargir davantage le domaine de la réponse. Sans cette magnification, les valeurs retournées risquent d'être très petites, parce que $\sin|\theta 1-\theta 2| \approx \sin|\theta 1-\theta 2| << 1$, lorsque les lignes sont parallèles.

Ainsi, l'écart entre les valeurs retournées par « ifdiff » devient trop étroite, ce qui rend plus difficile le travail de comparaison entre les couples. Étant donné que le sinus de l'angle peut être approximé par le même angle en radian pour un angle près de 0, nous convertissons simplement cet angle en degré en le multipliant avec 180/pi.

Suite à l'appel à la fonction « ifdiff », la fonction « Exist » met en mémoire l'indice de l'élément qui a la plus petite distance avec le couple (d,θ).

On arrive maintenant à l'étape d'insertion du couple (d,θ) dans l'accumulateur en se servant de l'indice retourné par la fonction Exist. La fonction à être employée est nommée « Insert ». Cette fonction a la responsabilité de déterminer si elle doit faire une mise à jour ou une simple insertion. Comme la fonction « Exist », la fonction « Insert » ne requiert pas la longueur du tableau à traiter, car cette information est contenue dans une variable (macro) globale. Son interface est la suivante :

```
int Insert(double d,double Theta,uint16 pos,THough *H_Acc) ;
```

où pos est la valeur retournée par la fonction « Exist ».

Cette fonction vérifie si l'indice pos se réfère au premier élément vide du tableau de Hough. Si tel est le cas, elle insère le couple (d,θ) à la position pos de l'accumulateur en initialisant le champ compteur de ce nouveau élément à 1. Cependant, si pos pointe à un élément qui existait déjà dans le tableau de Hough, alors il faudra appeler la fonction « Increment » dont l'interface est :

void Increment(double d,double Theta, uint16 pos,THough *H_Acc)

Cette fonction fait une mise à jour de l'élément de Hough qui existe déjà dans l'accumulateur et qui est pointé par l'indice pos (retourné par la fonction Exist) en tenant compte de la contribution de l'élément (d,θ) actuel selon une loi de la moyenne statistique suivante :

$$d = \frac{d + d_i \cdot cnt_i}{cnt_i + 1} \tag{4.28}$$

$$\theta = \frac{\theta + \theta_i \cdot cnt_i}{cnt_i + 1} \tag{4.29}$$

$$cnt_i = cnt_i + 1 \tag{4.30}$$

où (d,θ,cnt) sont définis dans la structure de Hough présentée plus tôt.

Après cette analyse, les couples (d,θ) vont tendre vers les valeurs stables avec leur champ compteur qui devient de plus en plus grand. Cette évidence ne serait pas vraie, s'il n'existait aucune droite évidente dans la zone d'analyse.

Les formules précédentes doivent être considérées plus particulièrement pour le cas traitant des droites horizontales. En fait, l'angle θ d'une droite horizontale peut être près de 0 ou de 180°. Lorsque θ_i (l'angle enregistré dans l'accumulateur de Hough) est près de 0°, pendant que θ (le nouveau angle) est près de 180°, peu importe la valeur du champ compteur, le nouveau θ, calculé selon la formule 4.29, va donner un résultat

invalide sur l'angle θ accumulé. Pour contourner ce problème, nous adoptons une convention particulière qui consiste à incrémenter tout simplement le champ compteur de l'élément i de 1 pour le cas des droites horizontales. Ce faisant, on estime que les angles près de 180° sont équivalents aux angles près de 0°. De toute évidence, la valeur d de l'élément à ajouter est presque identique à la valeur d_i car elles sont issues des droites horizontalement identiques. C'est pourquoi les deux autres champs de l'élément i restent inchangés.

Désormais, la taille du tableau de Hough est fixée par la variable globale hT*W1 et la longueur du tableau des coordonnées des coins par la variable globale cT*W1. Ces variables sont définies sous forme de macro dans le fichier « improces.h ». Par cette globalisation, tous les sous programmes se réfèrent automatiquement à ces quantités pour savoir la taille maximale des tableaux qu'ils ont à traiter. Intuitivement, on peut estimer que le nombre de lignes détectées dans une zone d'analyse est proportionnelle à la taille de cette zone. Cependant, on devrait être capable de détecter plus de lignes que de coins dans une fenêtre d'analyse donnée, c'est pourquoi la valeur hT est normalement choisie pour être supérieure ou égale à la valeur cT et les tailles de ces tableaux sont toutes fixées proportionnellement à W1.

Il peut arriver facilement que le tableau de Hough n'ait plus de place pour accepter un nouvel élément dans le cas où la fonction « Exist » retourne un indice égal à hT*W1. Il faut alors penser à une méthode pour contourner ce problème. À cet égard, il est irréaliste de penser à créer un tableau de Hough (et de coins) assez grand pour pouvoir contenir toutes les droites détectées, car le temps de calcul et de traitement croît proportionnellement avec la taille de l'accumulateur.

Heureusement, parmi ces droites, il y en a un certain nombre qui n'existe pas physiquement mais qui sont formées à cause de la lacune de notre méthode d'implantation. La figure 4.9 montre une scène d'analyse idéale où il n'existe pas de

pixels de nature bruitée. On peut voir très bien qu'il existe bien quatre lignes nettes qui se croisent entre elles pour former deux intersections.

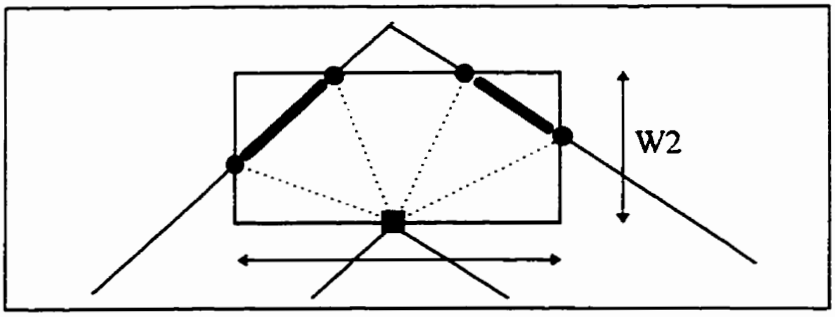


Figure 4.9 La lacune de notre méthode de construction de Hough

Imaginons que notre analyse se poursuit depuis un certain temps et actuellement notre pixel central est situé au petit carré noir. Ses voisins sont, selon notre convention, tous les pixels d'arêtes (les lignes noires épaisses) inclus dans cette fenêtre. En gardant le pixel central fixe, nous prenons un à un les pixels voisins pour calculer l'angle θ et la distance d des droites formées par ces couples de points. Un exemple de droites typiques sont présentées dans la figure 4.9 par les lignes pointillées. On sait très bien que dans ce cas, aucune ligne de ce type n'est valide, mais notre algorithme les considère quand même comme droites et les insère dans l'accumulateur de Hough. Évidemment, le champ compteur associé à ces fausses droites ne pourra jamais être élevé. Par conséquent, lorsque l'accumulateur vient à déborder, ce sont ces droites qui devront être les premières à être supprimées. Par convention, lorsque la fonction « Exist » nous retourne une valeur de pos égale à hT*W1 (le signe de débordement), nous devons réorganiser le tableau de Hough en essayant d'enlever tous les éléments ayant un champ compteur de valeur 1. Pour cela, il faut utiliser la fonction « reorganize » qui a de l'interface suivante :

```
void reorganize(THough * H_Acc, int &pos)
```

où la valeur retournée « pos » indique le nombre de l'élément de Hough qui reste après la réorganisation. C'est pourquoi la valeur pos se réfère souvent au premier élément vide restant dans l'accumulateur de Hough.

Une telle action peut être dite insuffisante lorsqu'il y a très peu d'éléments qui ont le champ compteur égal à 1. C'est pourquoi, il est plus efficace et plus rentable d'évaluer la prochaine valeur minimale du champ compteur des éléments restant en même temps que l'on balaie l'accumulateur de Hough pour faire le travail de réorganisation.

Après un boucle de balayage, nous évaluons le nombre d'éléments restant dans l'accumulateur de Hough pour voir s'il y a un risque de se faire remplir rapidement. Dans l'affirmative, un autre boucle de réorganisation serait préférable mais en éliminant cette fois-ci tous les éléments ayant un champ compteur égal à la valeur minimale du champ compteur des éléments de Hough restant. En continuant ainsi, nous pouvons être assurés que l'accumulateur de Hough sera bien réorganisé.

Toutefois, il y a un risque en faisant cela. En effet, lorsque la taille de tableau est petite, on doit refaire souvent l'action de réorganisation, ce qui empêche fortement les nouveaux éléments d'avoir la chance de croître. Par le fait même, on vient de tuer inconsciemment les bons candidats. Pour éviter un tel désastre, la taille convenable pour les tableaux de Hough est estimée à 140 éléments, tandis que la taille du tableau de coins est fixée approximativement à 70 éléments. Cependant, grâce à la performance de nos algorithmes, notre tableau de coins ne débordera jamais.

Revenons à notre exemple ci-dessus, on voit qu'il est inutile d'essayer de paramétrer les lignes droites passant par un tel pixel central, car ce pixel central cause fréquemment le débordement de l'accumulateur de Hough. Il vaut mieux l'abandonner et continuer notre analyse sur d'autres pixels centraux. On peut conclure qu'on est dans

cette situation, lorsque le nombre de fois qu'un pixel central courant cause le débordement du tableau de Hough dépasse une certaine limite. Dans notre cas, nous avons fixé cette fréquence de réorganisation « freq_org » à trois. On a évité de fixer « freq_org » à 1, car il peut arriver que le pixel central courant soit un bon candidat sauf qu'il est considéré un peu tard de telle sorte que l'accumulateur de Hough soit sur le point de déborder. C'est pourquoi, en lui donnant une chance, il va pouvoir rester en vie longuement, s'il est vraiment un bon candidat.

Bref, le technique de construction de l'accumulateur de Hough peut être résumé de la façon suivante :

```
/*Soit (r,c) la position en rangée et colonne du pixel en haut à gauche de la fenêtre d'analyse
(carré de 70x70)*/
Initialiser l'accumulateur de Hough
col←c
Pour rang : r+W2→r+W1-1
        Pour chaque pixel d'arête consécutif situant sur la rangée rang faire
                Déterminer la rangée et la colonne exacte de ce pixel central
                Trouver tous les pixels d'arêtes qui sont voisins du pixel central
                freq_org←0
                Pour chaque pixel voisin faire
                        Calculer θ entre pixel central et pixel voisin, formule (4.23)
                        Si θ < 0 alors
                                θ←θ+180°
                        Si θ==180° alors
                                θ←0
                        Si l'on est à la recherche des quatre coins extrêmes alors
                                Quantifier l'angle θ      (*)
                        Calculer d tenant compte de la translation d'axe, formule (4.24)
                        si l'accumulateur de Hough n'est pas plein alors
                                insérer ce couple dans l'accumulateur de Hough
                        sinon si c'est plein alors
                                reorganiser l'accumulateur de Hough, retouner l'indice de la
                                dernière place libre
```

```
                    insérer le couple(d,θ) dans l'accumulateur de Hough à cette
                    place libre
                    si freq_org >= seuil (3 dans notre cas) alors
                                abandonner le boucle <-> abandonner le point
                    sinon incrémenter freq_org de 1
                fin si
            fin pour chaque pixel voisin
        Fin pour chaque pixel central sur une rangée
Fin pour chaque rangée
```

4.9 REGROUPEMENT DES ÉLÉMENTS DE HOUGH

Nous avons vu que la fonction « Increment » est une fonction très efficace pour forcer les couples (d,θ) à tendre vers les valeurs fixes et très représentatives de l'ensemble des lignes qu'ils représentent. Plus il y a des points à analyser, plus les couples (d,θ) deviennent stables à moins qu'il n'existe aucun segment rectiligne dans la zone d'analyse.

Cependant, à cause de cette capacité « d'auto apprentissage », la fonction « Increment » peut créer des situations où plusieurs (d,θ) deviennent à la longue presque identiques, même s'ils sont initialement différents. Un tel phénomène peut arriver parce que ni la fonction « Exist » ni la fonction « ifdiff » ni la fonction « Increment » ont l'autorité de déplacer ou de supprimer quoi que ce soit dans l'accumulateur de Hough. Le mieux qu'elles peuvent faire, c'est de trouver l'élément dans l'accumulateur de Hough qui ressemble le plus à l'élément courant et faire un ajout adéquat. Il n'y a aucune garantie qu'un élément j, qui est différent de l'élément i pour l'instant, restera toujours différent jusqu'à la fin. Il faut donc penser à un algorithme qui regroupe davantage les (d,θ) après la construction de l'accumulateur de Hough.

Le regroupement des éléments de Hough réduit aussi le temps de calcul et d'analyse, si l'on tient compte du fait que pour la détection des quatre coins extrêmes d'une carte rectangulaire, il n'y a que les lignes verticales et horizontales qui sont importantes ; tandis que pour la détection des intersections de rues à l'intérieur de la carte, les seules lignes importantes sont celles qui sont parallèles deux à deux.

En tenant compte de ces trois cas, nous avons développé une procédure de regroupement ayant l'interface suivante :

void reduceHough(tableau de Hough, Option)

avec

Option == 0 pour le regroupement général des lignes identiques,

Option == MAP_INSIDE pour le regroupement qui élimine les lignes non-parallèles,

Option =/= MAP_INSIDE pour le regroupement qui élimine les lignes obliques.

4.9.1 Regroupement des éléments de Hough d'ordre général (Option ==0)

Un bon algorithme de regroupement des lignes identiques doit faire en sorte que chaque ligne soit représentée par un et un seul élément. Les autres éléments représentant cette même droite devront être regroupés. Évidemment, l'élément restant doit être le résultat de combinaison de tous les éléments représentant cette même droite. Autrement dit,

$$\text{nouveau } d = \text{fonction1}(d_1, d_2) \tag{4.31}$$

nouveau Theta=fonction2(θ_1,θ_2) (4.32)

Il est incorrect d'utiliser simplement un calcul de la moyenne pour réaliser les deux fonctions 1 et 2, car, pour le cas des droites horizontales ($\theta_1 \approx 180°$ et $\theta_2 \approx 0°$), la moyenne va donner un nouveau θ près de 90°, ce qui est complètement inacceptable.

C'est pourquoi, il faut vérifier, tout d'abord, si les lignes à regrouper sont horizontales, et dans l'affirmative, nous devons prendre la formule suivante pour déduire l'angle θ moyen :

$$\tilde{\theta} = \frac{\sin^{-1}(\sin(\theta_1)) + \sin^{-1}(\sin(\theta_2))}{2} \tag{4.33}$$

Pour les autres cas de θ, nous pouvons utiliser le calcul de la moyenne, c'est-à-dire :

$$\tilde{\theta} = \frac{(\theta_1) + (\theta_2)}{2} \tag{4.34}$$

Étant donnée que θ change, il est préférable que d soit modifié aussi. Encore là, on peut utiliser le calcul de la moyenne pour déterminer le nouveau d sauf pour le cas de deux lignes horizontales où les d sont essentiellement de mêmes grandeurs mais de signes contraires. Dans ce cas uniquement, il faut calculer la moyenne avec les d en valeur absolue.

4.9.2 Regroupement des éléments de Hough, méthode spécifique à la détection des lignes horizontales et verticales (Option =/= MAP_INSIDE)

Comme l'on a mentionné plus haut, pour détecter les quatre coins extrêmes d'une carte (supposée rectangulaire), il suffit d'éliminer toutes les lignes sauf celles horizontales et verticales. Il faut noter qu'à cette étape-ci, il existe encore dans l'accumulateur de Hough un bon nombre d'éléments issus de pixels bruités qui viennent nuire à l'interprétation de notre résultat. Un exemple de pixels bruités est illustré à la figure 4.10 où les pixels qui forment les lignes principales sont moins nombreux que les pixels bruités.

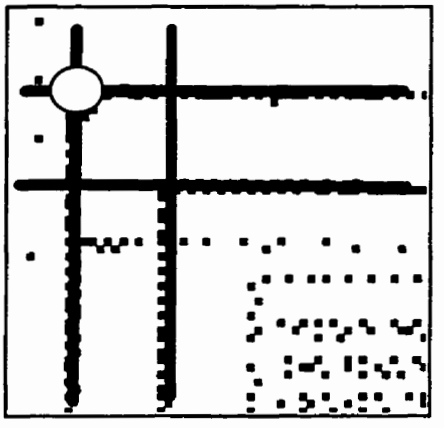


Figure 4.10 Différence entre les pixels bruités par rapport aux bons pixels. Seules les lignes de 0° et 90° (lignes épaises) sont essentielles pour déterminer les coins de la carte. Le coin extrême de la carte, pour cet exemple, est situé sur le centre du cercle. Les autres pixels n'appartenant pas à ces lignes principales sont considérés comme les pixels bruités.

Pour ne garder que les lignes horizontales et verticales, il suffit de balayer le tableau de Hough et vérifier si l'angle θ correspond à une droite horizontale ou verticale, dans l'affirmative, on passe à l'élément suivant, si non on supprime l'élément du tableau et poursuit le balayage.

De plus, pour augmenter nos chances d'obtenir des lignes de 0° et de 90° à cette étape-ci, nous avons fait de la quantification de l'angle θ avant même de calculer les d. L'action de quantification consiste à arrondir les valeurs de l'angle pour pouvoir ainsi dire qu'un angle entre [85 °.. 105°] peut être considéré comme un angle de 90°. Il est nécessaire de quantifier l'angle avant de calculer d, car la valeur de l'angle θ a beaucoup d'influence sur la valeur d. Dans notre cas, le pas de quantification utilisé est de 5°. Grâce à ces stratégies, nous sommes capables de détecter presque tous les coins extrêmes de la carte même dans la condition la plus mauvaise où les lignes obliques sont nombreuses. Cet algorithme ne sera prit en défaut qu'en présence d'une carte ayant les bordures trop usées, de telle sorte qu'il n'existe que peu de pixels d'arêtes.

Il est intéressant de remarquer que la méthode de quantification utilisée dans ce contexte, favorise le regroupement des angles de 90°, de 0° et de 180°, donc améliore la performance de la détection des lignes horizontales et verticales seulement. Par contre, cet algorithme ne peut pas être utilisé pour la détection des intersections de rues, car les lignes principales qui forment les intersections de rues ont des orientations très variées. Pour traiter une telle situation, il nous faut absolument un autre algorithme de regroupement.

4.9.3 Regroupement des éléments de Hough, méthode spécifique à la détection des lignes d'orientation diverse (Option == MAP_INSIDE)

La figure 4.11 illustre une intersection de rues typique dans laquelle on peut facilement remarquer que les lignes principales sont incomplètes tandis que les lignes bruitées sont abondantes.

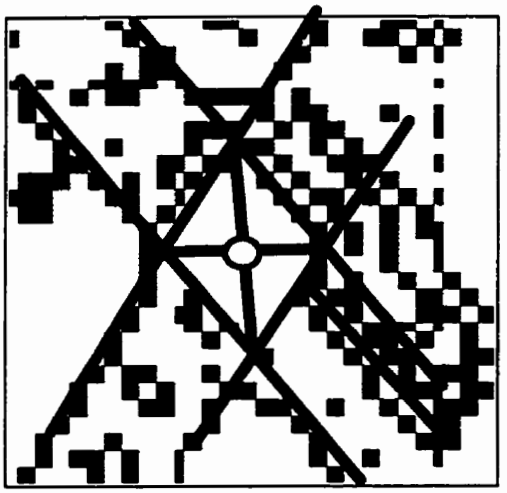


Figure 4.11 Différence entre les pixels bruités versus les bons pixels. Les lignes principales sont dans les directions 58° et 129°.L'intersection de rues se situe sur le centre du petit cercle. Les autres pixels n'appartenant pas à ces lignes sont considérés comme les pixels bruités (les angles sont tous exprimés par rapport à l'axe horizontal).

Malgré cela, nous avons essayé d'étudier des propriétés intrinsèques d'une image de rue et nous avons eu des conclusions suivantes (voir figure 4.12):

1. Toutes les grandes rues sommes composées de deux lignes parallèles séparées par une distance raisonnable (de 8 à 16 pixels),

2. Toutes les lignes qui composent la rue en question devraient avoir leur champ compteur élevé par rapport aux autres lignes (à moins que l'image contienne peu de pixels d'événement ou trop de bruit),

3. L'intersection de rue existe là où les lignes se croisent en angle de $90°\pm \xi$ et ce, couple par couple. Ainsi, le terme « croisé par couple » veut dire que si un couple est situé dans le premier plan du cercle géométrique, l'autre couple doit être nécessairement situé dans le plan complémentaire.

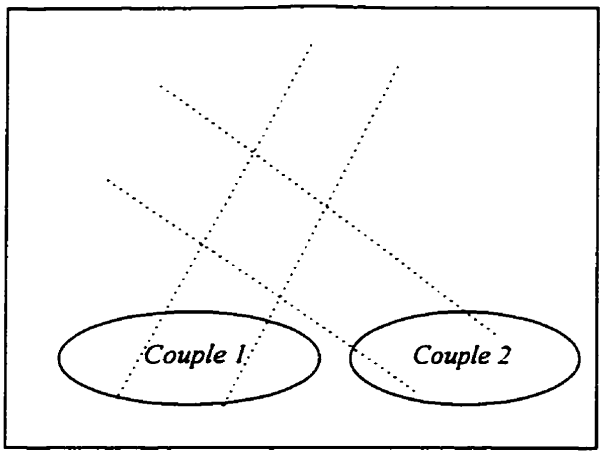


Figure 4.12 Les propriétés des lignes recherchées sont :

- ***Parallèles entre elles***
- ***Ayant un compteur relativement élevé***
- ***Croisées par couple***

Par conséquent, toutes les lignes qui ne sont parallèles à aucune des lignes sont certainement des lignes inutiles et devront être enlevées en premier. Après ce grand nettoyage, on se trouve avec un tableau de Hough plus simple, mais qui contient encore des lignes « bruitées », parce que celles-ci ont réussi à former un couple. Pour les éliminer, il suffit de remarquer qu'elles possèdent un champ compteur assez faible par rapport au champ compteur des lignes principales. Également, les lignes que l'on est intéressé à garder sont celles qui se croisent couple par couple à l'angle de 90^0 approximativement.

Nous avons, à cette étape, défini tous les termes nécessaires pour concevoir un pseudocode complet de regroupement. Supposons qu'on ait un tableau de Hough H bien rempli de longueur dynamique L, deux tampons M, N et un tableau de Hough secondaire Vectorpl qui va servir à emmagasiner temporairement le résultat du regroupement. La longueur L des tableaux est dynamique, car on va utiliser occasionnellement la fonction Delete pour enlever des éléments de Hough, ce qui entraîne un changement réel dans la taille effective des tableaux. Selon notre convention, l'information concernant la taille de l'accumulateur n'a pas besoin d'être mémorisée, car il est simple de la trouver juste en

balayant le tableau de Hough jusqu'à ce que l'on arrive au premier élément dont le champ compteur est nul.

Voici le pseudocode de regroupement dans le cas général :

```
m=0 ;
initialiser Vectorpl (Vecctorpl.cnt=0)
Pour i :0 → L (effetive)
        M← H[i]
        N.cnt=0
        Pour j :i+1 → L (effective)
                Si H[j] // M et leur distance en d doit dépasser un certain seuil (6 pixels dans
                notre cas) alors
                        Si H[j].cnt >= M.cnt OU N.cnt == 0 alors
                                N← M
                                M← H[j]
                        Sinon si H[j].cnt >= N.cnt alors
                                N← H[j]
                        Enlever cet élément j de Hough (désincrémenter j de 1 également)
                Fin si parallele
        Fin pour j
        Si N.cnt =/= 0 alors
                V[m++]← N
                V[m++]← M
        Fin si
Fin pour i
/*Cherchons les couples gagnant*/
Réinitialiser H (H.cnt=0)
Répéter
        Cherchons l'élément ayant le compteur le plus élevé (dans Vectorpl)
        Mémoriser le couple (lui et son camarade !)
        Supprimer ce couple du tableau
        Noter la distance entre ces deux lignes
Noter l'angle d'orientation de ces droites
Tant que l'on ne peut pas trouver un couple qui fait un angle évident avec  le couple (M,N) et
que V  n'est pas vide
```

Si V n'est pas vide alors /*V[0] et V[1] contient les deux droites les plus appropriées pour être utilisées avec M et N*/

V[2]←M

V[3]←N

V[4].cnt=0

Mémoriser les deux couples résultant dans H

FIN du pseudocode

4.10 DÉTERMINATION DES POINTS DE CROISEMENTS ENTRE LES ÉLÉMENTS DE HOUGH

À cette étape, l'accumulateur de Hough est sous une forme la plus propre et la plus concise (quatre éléments seulement qui ont les caractéristiques telles que présentées à la figure 4.12). Avec ces quatre éléments, nous devrions être en mesure de trouver quatre intersections. On peut remarquer que le centre de gravité de ces quatre points va donner une bonne estimation de l'intersection de rues recherchée.

Nous arrivons maintenant à l'étape de détermination des points d'intersections entre les éléments de Hough. On veut un algorithme le plus général possible de telle sorte qu'on n'ait pas besoin de distinguer si les lignes principales viennent du cas de détection des quatre coins extrêmes de la carte ou du cas de détection de point d'intersection de rues. Malgré cet objectif, il y a encore un cas de considération où il faut distinguer les deux types de détection. En effet, puisque nous voulons mettre le plus d'intelligence possible dans notre algorithme, nous avons exigé qu'un point de croisement dans le cas de détection des coins extrêmes devrait être entouré absolument par des pixels formant un segment horizontal et un segment vertical. Pour le cas de détection des intersections de rue, cette exigence ne s'applique pas car les intersections de rues en forme de T ne vérifient pas cette propriété. Ainsi lorsqu'on est dans le cas de détection des coins extrêmes, on doit utiliser la fonction « CntNeighbor » qui nous

retourne le nombre de pixels voisins dans les deux sens horizontaux et verticaux du pixel central. L'interface de cette fonction est :

int CntNeighbor(PICINFO In, int Rowc, int Colc, int Large, int Haut, int Option)

Cette fonction requiert comme entrées l'image sous la forme de représentation PICINFO, la position du pixel central en [Rangée, Colonne], la largeur et la hauteur de la fenêtre d'inspection et la nature de la détection. La fenêtre d'inspection définit la zone dans laquelle la fonction « CntNeighbor » compte le nombre des pixels voisins. La taille de cette fenêtre est variable, ce qui permet une plus grande utilisation. L'algorithme de la fonction « CntNeighbor » se résume comme suit :

Définir la taille et l'emplacement de la fenêtre d'inspection
Placer la fenêtre dans le sens horizontal d'abord
Cntbx←Compter le nombre de pixels d'événement
Placer la fenêtre dans le sens vertical
Cntby←Compter le nombre de pixels d'événement
Si Cntbx>0 ET Cntby>0 ET la différence entre Cntbx et Cntby est petite alors
    retourner(W2*(Cntbx+Cntby))
Sinon retourner ((Cntbx+Cntby)/3)

Ainsi, lorsque la fonction « CntNeighbor » retourne une petite valeur, ceci veut dire que le pixel central en question a très peu de voisin. Dans notre application, nous avons fixé la taille de la fenêtre d'inspection à 3x14 et le seuil de détermination pour « CntNeighbor » est fixé à 25 (équivalent à 3 pixels) pour la détection des coins extrêmes de carte. Pour le cas de détection des intersections de rues, nous n'avons pas utilisé cette fonction.

Peu importe le cas de détection, il faut éliminer les candidats qui n'ont aucun voisin. C'est pourquoi nous devons utiliser une fonction spécialisée pour le cas de détection des points d'intersection de rues. Cette fonction « isanEdge » a de l'interface suivante :

int isanEdge(uint32 Offset, FILE *EdgeFile)

La fonction « isanEdge » retourne la valeur 1 lorsque le point représenté par la valeur Offset existe ou est près en terme de [Rangée, Colonne] d'un des points emmagasinés dans EdgeFile. Nous n'avons pas besoin d'envoyer le paramètre Wpic ou In.w pour convertir Offset en des valeurs [Rangée, Colonne], car cette information (Wpic) peut être consultée dans EdgeFile.

De plus, lorsque plusieurs éléments se croisent approximativement à un même point, il y a de très forte probabilité que ce point soit une bonne intersection. Pour mesurer cette similitude, nous devons utiliser la fonction «labelcoin » dont l'interface est la suivante :

int labelcoin(uint32 Offset, uint16 w, intxv Wpic, coinINFO *coins)

où « w » représente le champ compteur correspondant au coin Offset. Si l'on réussit à associer ce nouveau coin à un des anciens coins, le champ compteur de l'ancien coin doit être mis à jour pour tenir compte de cet ajout et la fonction « labelcoin » retournera 0. Dans le cas où la fonction « labelcoin » ne peut associer le point referé par Offset à aucun des éléments enregistrés dans le tableau des coins, elle insère tout simplement ce nouveau candidat à la queue du tableau et elle retournera la valeur 1. La variable coins est le pointeur à ce tableau. Comme nous avons dit plutôt, la taille de ce tableau peut être fixé à 70. Chaque élément de ce tableau est de structure coinINFO qui contient les champs suivants :

```
struct coinINFO
{
    uint32 Offset;
    uint16 cnt;
};
```

On arrive maintenant à discuter l'implantation de la formule qui calcule le point d'intersection. Étant donné deux éléments de Hough H_1 et H_2, on peut calculer le point d'intersection (i,j) par les formules suivantes (voir section 3.2) :

$$\Delta = \sin(\theta_1 - \theta_2) \tag{4.35}$$

$$i = \frac{d_1 \cos(\theta_2) - d_2 \cos(\theta_1)}{\Delta} \tag{4.36a}$$

$$j = \frac{d_2 \sin(\theta_1) - d_1 \sin(\theta_2)}{\Delta} \tag{4.36b}$$

$$\text{poids} = \frac{cnt_1 + cnt_2}{2} \tag{4.37}$$

Lorsque les deux angles sont presque égaux, le déterminant Δ devient tellement près de 0 que les deux formules de i et j ne peuvent être utilisées. De toute manière, ce cas ne se produit jamais car les lignes parallèles qui peuvent rester « vivante » jusqu'à cette étape-ci sont sans doute des parallèles principales. Il n'est donc pas de question de calculer le point d'intersection entre ces parallèles. Puisque nous nous sommes intéressés uniquement au cas d'intersection à angle évident (l'angle d'intersection doit être entre 20° à 160°), nous ne rencontrons donc jamais le cas où Δ est près de 0.

Il faut noter que le point d'intersection (i,j) que l'on vient de calculer est exprimé par rapport à l'origine virtuelle (Row_Offset, Col_Offset). Ci-dessous est le pseudocode de l'algorithme complet de détermination des points d'intersection.

```
Pour i :0 → longueur effective de H_Acc
        Pour j : i+1 → longueur effective de H_Acc
                Déduire le point d'intersection entre H_Acc[i] et H_Acc[j]
                Si ce point est différent de (0,0) et que le point d'intersection se situe dans la
                fenêtre d'analyse ALORS
                        champ compteur de coin←(champ compteur de H_Acc[i] + champ
                        compteur de H_Acc[j])/2
                        Vérifier si ce point existe physiquement dans EdgeFile /*ne pas oublier
                        la translation d'axe*/
```

```
                    Si oui ALORS
                        Enregistrer ce coin /*par simple insertion ou mise à jour*/
                    Fin Si
                Fin Si
            Fin pour j
        Fin pour i
```

Suite à l'utilisation de cet algorithme, nous obtenons un tableau contenant des coins les plus probables. Ils sont au nombre de quatre, s'il s'agit du cas de détection de point d'intersections de rues, et ils sont au nombre de N s'il s'agit du cas de détection des coins extrêmes de la carte, où N est un nombre fini. Si nous supposons que les intersections de rues sont issues des grandes rues (par exemple Boulevard ou Avenue) de telle sorte qu'elles pourront être représentées par deux lignes principales parallèles séparées par une largeur de 8 à 15 pixels, nous pouvons aller à une étape plus loin qui consiste à calculer le centre de gravité de ces coins et l'estimer comme le point d'intersection de rues. Dans le cas de la détection des quatre coins extrêmes de la carte, il ne faut pas calculer le centre de gravité pour obtenir le résultat recherché. Il faut donc une étape d'interprétation finale sur les points de croisements.

4.11 INTERPRÉTATION FINALE SUR LES POINTS DE CROISEMENTS

Tous les algorithmes et les fonctions discutés plus haut sont utilisés entièrement dans la fonction « CornerDetect » dont l'interface est la suivante :

coinINFO CornerDetect(uint16 Row, uint16 Col, PICINFO &In, int Verbose, int Option)

où

Row est le centre de la fenêtre d'analyse en terme de Rangée,

Col est le centre de la fenêtre d'analyse en terme de Colonne,

In est le contenu de l'image selon la structure PICINFO de xv,

Verbose (égale à 1) pour activer l'affichage des résultats intermédiaires (en cas de vérification du cheminement d'exécution du programme),

Option est la variable qui fait la différence entre le cas de détection des quatre coins extrêmes de la carte et le cas de détection de l'intersection de rue.

Pour éviter les cas de traitement des frontières où le point [Row,Col] donné se situe trop près des bordures de l'image de telle sorte la fenêtre d'analyse correspondant risque de toucher des régions qui se situent en dehors de l'image, nous devons à chaque fois vérifier ces conditions avant d'aller plus loin dans la fonction "CornerDetects". L'algorithme de traitement des cas frontaliers se résume en quelques lignes suivantes:

```
Si Rangée+W1/2>Hauteur de l'image ALORS
        Rangée=Hauteur de l'image-W1/2
Si Rangée<W1/2 Alors
        Rangée=W1/2
Si Colonne+W1/2>Largeur de l'image Alors
        Colonne=Largeur de l'image-W1/2
Si Colonne<W1/2 Alors
        Colonne=W1/2
```

Évidemment, cet algorithme ne s'applique qu'à des images ayant la largeur plus grande que la largeur de la fenêtre d'analyse (W1=70) et l'hauteur plus grande que l'hauteur de la fenêtre d'analyse (W1=70). Or, cette condition est toujours respectée en cas d'image des cartes géographiques. La fonction « CornerDetect » ne retourne qu'un seul élément qui représente la position de l'intersection détectée en terme de {Offset,

valeur de compteur}. Dans le cas où elle n'arrive à détecter aucun point, elle retournera {0,0}.

Autrement dit, pour détecter les quatre coins extrêmes de la carte, il faut appeler cette fonction quatre fois avec quatre positions centrales différentes. C'est pourquoi, nous avons défini cinq valeurs possibles pour Option :

MAP_CORNER0	1	/*coin extrême en bas à gauche */
MAP_CORNER1	2	/*coin extrême en haut à gauche */
MAP_CORNER0	3	/*coin extrême en haut à droite */
MAP_CORNER0	4	/*coin extrême en bas à droite */
MAP_INSIDE	5	/*coin de l'intersection à l'intérieur de l'image */

Grâce à l'information sur la valeur de la variable Option, on peut orienter facilement nos stratégies d'interprétation. En effet, lorsque la variable Option est différente de la valeur MAP_INSIDE, on peut dire tout de suite qu'il s'agit du problème de détection d'un des quatre coins extrêmes de la carte. Pour la détection d'un de ces coins, il faut prévoir quatre cas (quatre types de MAP_CORNER) parce que chaque cas exige que le coin extrême soit situé à des endroits différents. Par exemple, pour le coin C1 (en haut à gauche), il faut que l'on sélectionne parmi les éléments du tableau de coins le point situant le plus en haut et le plus à gauche. Ainsi l'algorithme d'interprétation des coins extrêmes de la carte est comme suit :

MINROW←la hauteur de l'image-2
/*C'est la valeur maximale que la variable Rangée peut prendre*/
MINCOL← la largeur de l'image-2
MAXROW=MAXCOL←2
Pour i :0→ la taille effective maximale du tableau de coins FAIRE

```
        Déterminer la position en terme de [Rangée, Colonne]
        Si la valeur du champ compteur de l'élément actuel est >=seuil_MaxCntCoin ET le
        nombre de pixels voisins est supérieur à 7 pour la fenêtre d'analyse 2x10 ALORS
                Si Option == MAP_CORNER0 ALORS
                        Si Colonne <= MINCOL+2 ET Rangée >= MAXROW-2 ALORS
                                MINCOL=Colonne
                                MAXROW=Rangée
                Si Option == MAP_CORNER1 ALORS
                        Si Colonne <= MINCOL+2 ET Rangée <= MINROW+2 ALORS
                                MINCOL=Colonne
                                MINROW=Rangée
                Si Option == MAP_CORNER2 ALORS
                        Si Colonne >= MAXCOL-2 ET Rangée <= MINROW+2 ALORS
                                MAXCOL=Colonne
                                MINROW=Rangée
                Si Option == MAP_CORNER3 ALORS
                        Si Colonne >= MAXCOL-2 ET Rangée >= MAXROW-2 ALORS
                                MAXCOL=Colonne
                                MAXROW=Rangée
        Fin si champ compteur
Fin pour i
cas Option égalet à
        MAP_CORNER0 alors Réponse←MINCOL+MAXROW*largeur de l'image
        MAP_CORNER1 alors Réponse←MINCOL+MINROW*largeur de l'image
        MAP_CORNER2 alors Réponse←MAXCOL+MINROW*largeur de l'image
        MAP_CORNER3 alors Réponse←MAXCOL+MAXROW*largeur de l'image
Fin cas Option
```

Cet algorithme est implanté dans la fonction « coinmaxencnt » qui a l'interface suivante :

coinINFO coinmaxencnt(coinINFO *coins, PICINFO In, int Option, float MaxCntCoin).

Il est intéressant de mentionner la stratégie que nous avons utilisé pour extraire les valeurs extrêmes en rangée et en colonne d'un coin donné. Prenons par exemple le

cas de détection du coin C1 (en haut à gauche). Soient 2 points potentiels (30,23) et (31,10) et supposons que la vraie position du coin se situe à (30,10). Si l'on compare la rangée et la colonne des coins potentiels avec MINROW (sans le terme +2) et MINCOL (sans le terme +2) respectivement, on va devoir refuser le point (31,10) parce que ce point correspond à une rangée plus basse que MINROW, pourtant ce point ne fait qu'une petite distance avec la vraie position. Pour éviter un tel problème, nous avons atténué les valeurs de MAXROW et de MAXCOL (de 2) et accentué les valeurs de MINROW et de MINCOL de 2 lors des comparaisons.

Dans cet algorithme, nous avons introduit un autre seuil (MaxCntCoin) pour sélectionner les bons coins à traiter car on ne veut pas que tous les coins (peu importe leur champ compteur) participent à cette sélection. Le seuil est variable d'une séance d'interprétation à une autre et est déterminé par la fonction appelant « CornerDetect ».

Étant donné que le champ compteur de chaque coin est égal à la moyenne des champs compteurs de deux lignes qui forment ce coin (voir algorithme section 4.10), aucun coin ne peut avoir un champ compteur plus élevé que le champ compteur maximal des éléments de Hough (MaxCntHough). Évidemment, si la fonction « CornerDetect » appelle la fonction « coinmaxencnt » avec un seuil égal à MaxCntHough, aucun coin ne sera accepté. Par contre, si la fonction « CornerDetect » appelle la fonction « coinmaxencnt » avec un seuil égal à la moitié de MaxCntHough, tous les coins seront considérés, même les candidats les plus indésirables. Il est donc préférable de diminuer graduellement le seuil MaxCntCoin qui doit être situé entre [MaxCntHough, MaxCntHough/2]. Pour notre application nous avons fait varier MaxCntCoin en trois phase [MaxCntHough, MaxCntHough/1.5, MaxCntHough/2]. La fonction « CornerDetect » termine ses opérations lorsque la fonction « coinmaxencnt » lui retourne un coin. Comme on va voir dans le prochain chapitre, le résultat de sélection utilisant ce seuil est très convainquant.

Pour le cas de détection des points d'intersection de rues, il s'agit tout simplement de calculer le centre de gravité des quatre coins fournis par le tableau de coins. Ces formules sont :

$$i = \frac{\sum_1^4 c_{kx}}{4} \tag{4.38}$$

$$j = \frac{\sum_1^4 c_{ky}}{4} \tag{4.39}$$

où ix et iy désignent le composante en x et en y du coin k respectivement.

Dans le cas où la région donnée ne contient aucun coin détectable, ou bien dans le cas où cette région contient trop de bruit de telle sorte que la fonction « CornerDetect » n'arrive pas à sélectionner un seul candidat, elle retournera {0,0}. Même dans le cas où la fonction « CornerDetect » retourne un candidat unique, nous ne nous sommes pas arrêtés là. Pour être sûr, nous répétons la détection de point d'intersection avec une nouvelle fenêtre ayant des centres différents à chaque fois.

La règle de détermination de l'emplacement de la nouvelle fenêtre d'analyse n'est pas unique. Lorsque la fonction « CornerDetect » nous retourne un point avec [Rangée, Colonne] différent de [Rangée, Colonne] initial, il y a toujours un risque de déplacer la fenêtre d'analyse à ce nouveau point, car ce point peut être formé à partir des lignes bruitées. Par contre, il est totalement inacceptable de déplacer la fenêtre d'analyse dans le sens contraire. La seule solution acceptable est de déplacer la fenêtre d'analyse avec un très petit pas vers la direction où se situe le nouveau point [Rangée, Colonne]. Ce pas est :

$$pas_y = \frac{Ancienne\_Rangee - Nouvelle\_Rangee}{4} + perturbation\_y \qquad (4.40)$$

$$pas_x = \frac{Ancienne\_Colonne - Nouvelle\_Colonne}{4} + perturbation\_x \qquad (4.41)$$

où les variables perturbation_y et perturbation_x sont les éléments de perturbation qui ajoutent un petit bruit dans le sens de déplacement de la fenêtre d'analyse. Le nouvel élément retourné par « CornerDetect » devrait être inséré dans un tableau de coin selon les techniques de regroupement que nous avons discuté précédemment (fonction « labelcoin »). Le nouveau point [Nouvelle_Rangée, Nouvelle_Colonne] utilisé dans les formules 4.40 et 4.41 devrait avoir un champ de compteur le plus élevé parmi les coins enregistrés dans le tableau de coins. C'est pourquoi, nous devons trier le tableau de coins en utilisant « qsort » avant d'utiliser les formules 4.40 et 4.41 pour la prochaine itération.

Nous répétons cette opération cinq fois jusqu'à ce qu'une des situations suivantes soient arrivées :

1.Le nombre de fois limite (5) est atteint OU,

2.« CornerDetect » détecte M fois le même coin.

Le résultat final de la détection est le coin qui a un champ compteur le plus élevé parmi les coins emmagasinés dans le tableau de coins.

Nous venons de voir que pour détecter une intersection de rue, nous devons appeler la fonction « Cornerdetect » en lui fournissant un point de départ [Rangée, Colonne]. Que faire si nous voulons détecter la position des coins extrêmes d'une carte, alors que nous ne savons pas a priori avec quel point de départ nous devons fournir. Il est impensable de deviner ces valeurs en espérant que la fonction « CornerDetect » nous

retournera une valeur non nulle, car le temps que la fonction « CornerDetect » prenne pour faire ces démarches est considérable. Pour résoudre ce problème, on a développé une autre fonction ayant le nom « LocateCorner » dont l'interface est comme suit :

coinINFO LocateCorner(PICINFO In,int dirr, int dirc, int Verbose)

Cette fonction « lit » le contenu de l'image In dans la direction spécifiée par les variables directionnelles dirr (pour la direction verticale) et dirc (pour la direction horizontale). Cette fonction réalise principalement les deux étapes suivantes :

1.trouver la position approximative de la région dans laquelle se trouve le coin extrême que l'on recherche et,

2.lancer la procédure « CornerDetect » utilisant comme position de départ la position graphique qu'elle vient de trouver.

Cette fonction retournera la position du coin détecté selon le format de la structure coinINFO. Elle doit être utilisée quatre fois pour détecter les quatre coins extrêmes de la carte. Elle est généralisée pour la détection de tous les quatre coins, car dans tous les cas, cette fonction déplace la fenêtre d'analyse dans les directions différentes selon les mêmes formules suivantes :

$$\text{Rangée} = \text{Rangée} + \text{dirr} * \text{RowStep} \qquad (4.42)$$

$$\text{Colonne} = \text{Colonne} + \text{dirc} * \text{ColStep} \qquad (4.42)$$

Les variables directionnelles ont des amplitudes fixées à 1 en valeur absolue. Leur signe détermine le type du coin recherché. Par exemple, lorsque dirr=1 et dirc=1, les variables Rangée et Colonne croissent toujours, ce qui est équivalent à déplacer la fenêtre d'analyse de haut en bas (Rangée croissante) et de gauche à droite (Colonne

croissante), ce qui correspond à la situation de détection du coin extrême en haut à gauche (C1), car c'est le seul cas où on peut déplacer la fenêtre d'analyse dans une telle direction pour trouver la région susceptible d'englober le coin à rechercher. Ainsi, pour détecter le coin C0, il faut commencer la fenêtre d'analyse au point (Rangée=Hauteur de l'image-W1, Colonne=0) et déplacer cette fenêtre dans la direction de dirr=-1 et dirc=1. Pour détecter le coin C2, il faut commencer la fenêtre d'analyse au point (Rangée=0, Colonne=Largeur de l'image-W1) et déplacer cette fenêtre dans la direction de dirr=1 et dirc=-1. Finalement, pour détecter le coin C3, il faut commencer la fenêtre d'analyse au point (Rangée=Hauteur de l'image-W1, Colonne=Largeur de l'image-W1) et déplacer cette fenêtre dans la direction de dirr=-1 et dirc=-1.

Le pas d'avancement de la fenêtre d'analyse est contrôlé par la valeur Colstep dans la direction horizontale et par la valeur Rowstep dans la direction verticale. Ces valeurs sont choisies de telles sorte que la fenêtre d'analyse déplace toujours vers le centre de l'image et que la fenêtre d'analyse, après déplacement, recouvre en partie la surface recouvert par la fenêtre d'analyse actuelle. Ainsi, la méthode de détermination de Rowstep et de Colstep est comme suit :

$$\text{ratio} = \frac{\text{Largeur de l'image}}{\text{Hauteur de l'image}} \tag{4.43}$$

$$\text{Colstep} = W1 - 2 \cdot W2 \quad \text{et} \quad \text{Rowstep} = \frac{\text{Colstep}}{\text{ratio}} \quad \text{si ratio} > 1 \tag{4.44}$$

$$\text{Rowstep} = W1 - 2 \cdot W2 \quad \text{et} \quad \text{Colstep} = \frac{\text{Rowlstep}}{\text{ratio}} \quad \text{si ratio} < 1 \tag{4.45}$$

La quantité (W1 - 2*W2) nous assure que la prochaine fenêtre recouvre toujours une partie de la surface recouvert par la fenêtre actuelle. À chaque fenêtre donnée, la fonction « LocateCorner » lance un appel à la fonction « CntNeighbor » en lui fournissant comme point central le point ayant des coordonnées [Rangée+W1/2, Colonne+W1/2] où W1 est la largeur de la fenêtre d'analyse. Cette fonction retourne le

nombre de pixels qui se trouve dans la région de dimension (W1-W2)X(W1-W2) qui entoure le point central. Lorsque ce nombre n'est pas assez élevé (cinq pixels), on continue à déplacer la fenêtre d'analyse. La figure 4.13 explique les procédures de détermination de la région susceptible de contenir le coin extrême en haut à gauche d'une carte.

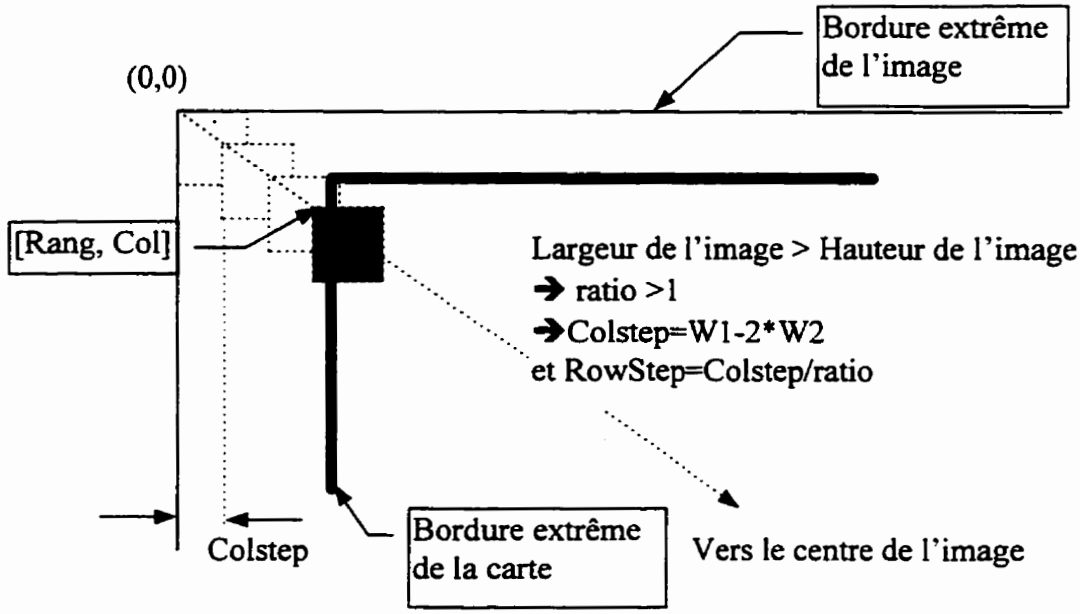


Figure 4.13 Procédures de détermination des pas d'avancement de la fenêtre d'analyse (carrés gris)

Effectivement, lorsque la largeur de l'image est plus importante que la longueur de l'image, il faut que le pas d'avancement dans le sens horizontal soit plus prononcé que le pas d'avancement dans le sens vertical, d'où Colstep>Rowstep. Les carrés gris dans la figure 4.13 montre les déplacements itératifs de la fenêtre d'analyse. On arête le déplacement de cette fenêtre lorsqu'elle englobe une région (carré gris foncé) qui contient suffisamment de pixels d'événement.

La figure 4.13 est un exemple où la fenêtre d'analyse rencontre la bordure verticale de carte en premier. Il arrive parfois qu'une carte géographique soit posée de telle sorte que la fenêtre d'analyse rencontre la bordure horizontale d'abord. Par conséquent, lorsque la fonction « LocateCorner » arrive à un endroit où il commence à

avoir des pixels d'événements, on est dans une des deux situations suivantes (voir figure 4.14)

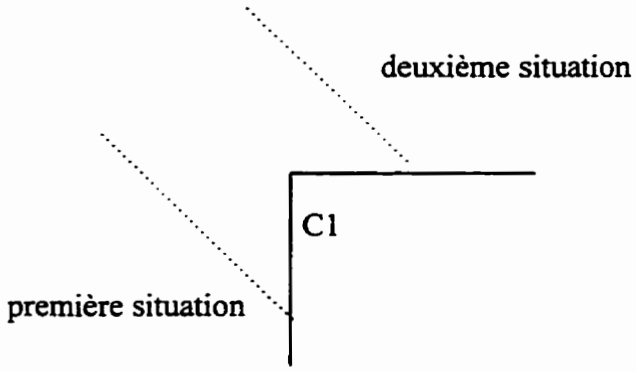


Figure 4.14 Les deux situations que la fonction « LocateCorner » rencontre lorsqu'elle recherche les bordures de la carte

À ce moment, on mémorise le point [Rang, Col] de la dernière fenêtre d'analyse et on essaie ensuite de déplacer verticalement la fenêtre d'analyse en fixant la valeur Col et en faisant diminuer (ou augmenter) graduellement la valeur Rang selon l'équation :

$$Row = Row - dirr^* RowStep \tag{4.46}$$

On laisse la fonction « LocateCorner » de continuer à faire ainsi jusqu'à ce que la fenêtre d'analyse atteint une région qui ne contient plus de pixel d'arêtes, auquel cas on vient de quitter la dernière bordure horizontale. De toute évidence, la fenêtre d'analyse précédente devrait contenir la bordure horizontale et probablement le coin extrême à rechercher si et seulement si l'on est dans la première situation. Cependant, si l'on est dans la deuxième situation, il est évident que la fonction « LocateCorner » va attraper une région vide aussitôt qu'on essaie de déplacer la fenêtre d'analyse verticalement. Par cette procédure, on est en mesure de savoir si l'on est dans la première situation ou non. Lorsqu'on est dans la deuxième situation, on doit recommencer la recherche en faisant cette fois-ci varier Col et en fixant la valeur de rangée à la valeur Rang trouvée plus haut.

En déplaçant la fenêtre horizontalement, la fonction « LocateCorner » devrait croiser sûrement l'extrémité recherchée. On peut maintenant appeler la fonction « CornerDetect » pour trouver le coin extrême en utilisant comme point de départ le point [Rang,Col] de la dernière fenêtre d'analyse.

4.12 CONCLUSIONS

Les algorithmes que nous avons décrits dans ce chapitre sont de type ad-hoc. Ils ne sont pas nécessairement les meilleurs détecteurs de coins, de façon générale. Nous les avons développé en nous servant des propriétés intrinsèques des coins de rues dans une image. Nous avons également établi des contraintes telles que la carte doit avoir quatre coins extrêmes visibles et que les rues à traiter doivent avoir une largeur entre 8 et 15 pixels. Plus particulièrement, la base de données BDR a été crée pour la fin de simulation. Pour les applications réelles, les usagers ou les opérateurs pourraient avoir leur propre façon de créer BDR, mais il faut que les paramètres essentiels suivants y soient définis, notamment le dpi et l'échelle de la carte. Ces deux données sont particulières pour chaque carte et elles sont indispensables pour évaluer la distorsion globale de la carte.

Il y a un intérêt à corriger la distorsion globale de l'image avant de commencer à aligner les points à l'intérieure de la carte. À l'ouverture d'une nouvelle carte, on doit commencer par trouver les quatre coins extrêmes de la carte et déterminer les positions de destinations pour ces quatre coins (en fait, on veut s'assurer que la largeur et la longueur des bordures horizontales et verticales soient égales). En fournissant les points de source et de destination de ces quatre coins extrêmes à l'algorithme d'alignement conçu par l'équipe du CRIM, nous obtiendrons une carte globalement réalignée dans laquelle les erreurs de distorsions sont faibles. Pour la prochaine étape de réalignement local, nous pouvons nous servir de cette carte partiellement réalignée pour détecter les

points de contrôle. Comme nous pouvons constater, nous avons essayé de concevoir les algorithmes de façon qu'ils soient généraux. Toutefois, il faut à un moment donné réduire ces réponses en une seule. En faisant cela, nous prenons le risque de faire échouer nos algorithmes, lorsque il n'existe pas de rues ayant les trois caractéristiques espérées (section 4.9). Habituellement, il est plus préférable de détecter les intersections des grandes rues, car ces rues ont des largeurs intéressantes. Cependant, si l'échelle de la carte était trop grande de telle sorte que même un grand boulevard est à peine visible dans l'image, nos algorithmes seraient probablement échoués. Nous verrons dans le prochain chapitre la structure générale de notre programme ainsi que la performance de nos algorithmes face à des situations diverses qu'une image réelle peut avoir.

Chapitre 5

Analyse de performance

5.1 OBJECTIF

Nous avons vu en détail dans les chapitres précédents les outils utilisés pour trouver les points d'intersection de rues. Ces principaux outils sont le modèle de Morrone, la transformée de Hough, la méthode de triangulation, quelques formules mathématiques simples permettant de créer les données pour nos simulations et quelques stratégies pour interpréter les résultats. Nous avons également expliqué les méthodes d'implantation logicielle de ces algorithmes. Pour que ces implantations soient utilisables, il nous faut également développer des interfaces qui contrôlent la séquence d'utilisation de ces implantations.

Dans ce chapitre, nous décrivons la structure générale de notre programme pour déterminer les interfaces dont nous devons créer pour que nos algorithmes soient utilisables par l'équipe du CRIM. En suite, nous évaluerons la performance de nos algorithmes en ce qui concerne la détection des coins extrêmes des cartes réelles (27

images) et la détection des intersections de rues dans une région donnée à partir des deux cartes réelles. Finalement, nous illustrerons le résultat d'alignement en utilisant le module Aligner développé par l'équipe du CRIM. Nous allons commencer d'abord par la description de la structure générale de notre programme.

5.2 LA STRUCTURE GÉNÉRALE DU PROGRAMME

Dans le chapitre 4, nous avons décrit tous les outils et les paramètres dont on a besoin pour faire de la détection des coins extrêmes de carte ainsi que de la détection des points d'intersection de rues. Dans cette section, nous présentons toutes ces fonctions en les intégrant ensemble. La figure 5.1 montre l'organigramme de la procédure de détecteur de coin, la fonction « CornerDetect ».

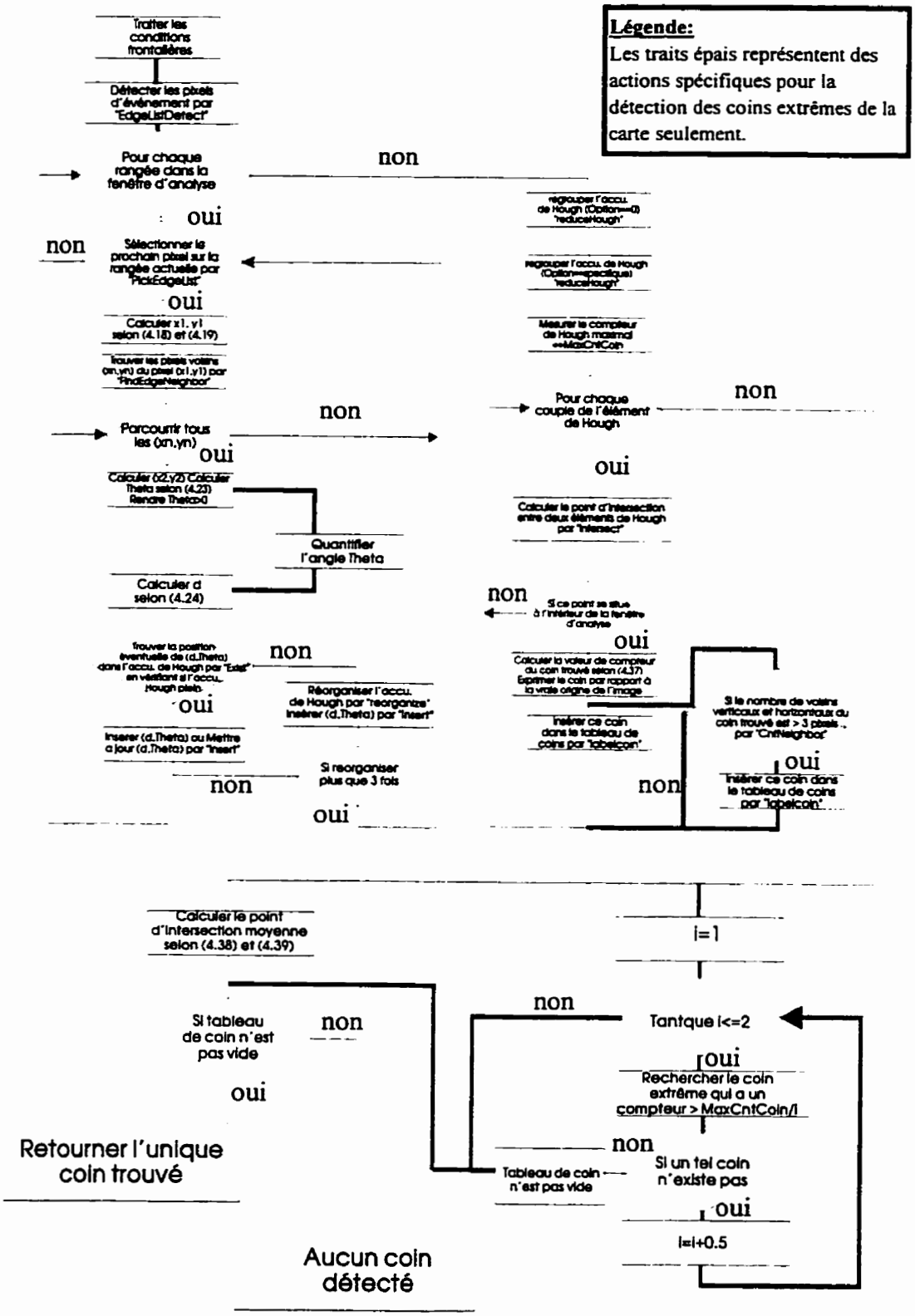


Figure 5.1 ***Organigramme de la fonction « CornerDetects »***

Comme nous avons discuté précédemment, cette procédure est utilisée pour détecter les intersections de rues ainsi que les coins extrêmes de carte, à condition de bien préciser le type de coin dont on veut détecter et de bien fournir un point de référence. On a vu également que pour détecter les coins extrêmes de carte, il faut utiliser la fonction « LocateCorner » pour positionner correctement la fenêtre d'analyse avant d'appeler la fonction « CornerDetect ». Ainsi, pour détecter les quatre coins extrêmes de carte, il faut répéter quatre fois ces deux opérations. Tandis que pour détecter une intersection de rues, il faut utiliser la fonction « CornerDetect » en un certain nombre de fois jusqu'à ce que la réponse reste stable ou jusqu'à ce que le nombre de fois limite soit atteint. Malgré que les deux types de coins utilisent la même fonction « CornerDetect », ils requièrent des opérations totalement différentes au départ. C'est pourquoi, nous avons implanté deux fonctions d'interface supplémentaires: la fonction "detect4corners" et la fonction "detectcorners".

5.2 LES INTERFACES SUPPLÉMENTAIRES

La première fonction d'interface que nous voulons parler est la fonction qui contrôle les séquences d'opérations en vue de détection des quatre coins extrêmes de carte. L'interface de cette fonction est :

```
int detect4corners(PICINFO In) ;
```

Tout ce que cette fonction a besoin comme entrée est l'image source emmagasinée selon la structure PICINFO. Cette fonction retourne quatre coordonnées graphiques des quatre coins extrêmes de la carte au biais de la variable

```
extern Ref_CtrlPtr ref_point[];
```

Cette variable, qui représente un tableau de quatre éléments, est déclarée dans le système global de l'équipe du CRIM. La fonction « detect4corners » affecte ses résultats de détection du coin extrême i aux champs x,y de l'élément i de ref_point selon la convention introduite à la section 1.2. On a fait en sorte que cette fonction soit la plus robuste et efficace possible pour minimiser les erreurs de détection. En bref, cette fonction suit l'algorithme suivant :

```
Algorithme de la fonction de detect4corners pour une image de largeur W et de hauteur H :
        Localiser le coin en haut à gauche
        /*commence par (0,0) et incrémenter graduellement en rangée et en colonne*/
        Si un tel coin est détecté ALORS
                Affecter ce résultat aux champs ref_point[1].x et ref_point[1].y ;
        Localiser le coin en bas à droite
        /*commence par (H,W) et décrémenter graduellement en rangée et en colonne*/
        Si un tel coin est détecté ALORS
                Affecter ce résultat aux champs ref_point[3].x et ref_point[3].y ;
        Localiser le coin en bas à gauche
        /*commence par (H,0) et décrémenter graduellement en rangée et incrémenter en colonne*/
        Si un tel coin est détecté ALORS
                Affecter ce résultat aux champs ref_point[0].x et ref_point[0].y ;
        Localiser le coin en haut à droite
        /*commence par (0,W) et incrémenter graduellement en rangée et décrémenter en colonne*/
        Si un tel coin est détecté ALORS
                Affecter ce résultat aux champs ref_point[2].x et ref_point[2].y ;
```

Lorsque la fonction "detect4corners" n'arrive pas à détecter un coin en particulier, les champs x et y de l'élément ref_point correspondant ne seront pas affectés. Pour contrôler ces changements, nous nous entendons avec l'équipe du CRIM de remettre à 0 les champs x,y,lon,lat des quatre éléments de ref_point à chaque fois qu'on ouvre une image. Par conséquent, les champs qui restent inchangés après l'utilisation de la fonction "detect4corners" signifient que cette fonction n'arrive pas à détecter les coordonnées graphiques du coin en question.

Tenant compte de l'importance de ces quatre points de référence, nous avons entendu avec l'équipe du CRIM de permettre aux usagers (opérateurs) de corriger manuellement les résultats de détection des quatre coins extrêmes de carte par l'interface décrite à la section 4.3.3.2. Évidemment, lorsque la disposition des coins extrêmes n'est

pas trop « anormale » (polygone formée par ces quatre coins n'est pas trop distordue) nous pouvons passer à la prochaine étape qui consiste à déduire les formules de conversion et à détecter un ensemble fini des intersections de rues sélectionnées dans le fichier « bdg » afin de préparer les données essentielles pour l'algorithme d'alignement. Lorsqu'on n'arrive pas à détecter correctement tous les quatre coins extrêmes de carte, on ne peut plus continuer, car on n'est pas capable de déduire les formules de conversion. Dans ce cas, on peut signaler les opérateurs de rétablir cette anomalie en temps différé en rapportant cet événement dans un fichier de contrôle que les opérateurs vont examiner à un moment donné ou en temps réel en avertissant les opérateurs de faire des démarches nécessaires.

Pour atteindre un degré d'automatisme élevé, il faut que les formules de conversion soient reconsidérées à chaque fois que nous traitons une nouvelle image. Pour cela, il suffit de forcer les {A, B, C, D} (définis à la section 4.3) et les 4 éléments du tableau ref_point à devenir 0. Étant donné que nous ne pouvons pas contrôler le moment où le programme principal, développé par l'équipe du CRIM, ouvre ou ferme une image, nous avons décidé de laisser à l'équipe du CRIM la responsabilité de remettre à 0 les contenus de ref_point. Idéalement, nous voulons que l'utilisateur soit capable de voir afficher sur l'écran les informations concernant les coordonnées géographiques et graphiques de tous points de l'image aussitôt qu'il ouvre une nouvelle image. Cela implique que l'équipe du CRIM devrait réaliser les opérations suivantes après l'ouverture d'une image.

```
Initialiser les ref_point à 0,
Appeler la fonction int getref(PICINFO Inimage, char* bdr_file)
Si « getref » retourne Échec ALORS
        Permmettre à l'usager de faire des corrections nécessaires
        Si correction incomplète ALORS
                Fin des opérations, effacer {A, B, C, D} et contenus de ref_point
        Fin Si
Fin Si
```

Les opérations précédentes ne sont qu'à titre de suggestions, car elles ne peuvent pas être réalisées par nous. Tout ce qu'on doit faire, c'est de préparer la fonction « getref » qui a été implantée selon l'algorithme suivant :

```
Remettre à 0 les A, B, C, D
Appeler la fonction "detect4corners" en lui transférant le paramètre Inimage
Si on a détecté avec succès la position graphique des quatre coins extrêmes alors
        Lire bdr et calculer A, B, C, D éventuellement (par la fonction « getdbinfo »)
        Si « getdbinfo » retourne SUCCES Alors
                retourner SUCCES
        Sinon retourner ECHEC
        Fin Si
Sinon retourner Échec
FinSi
```

Une fois que les formules de conversion sont bien définies, nous aurons rempli toutes les conditions nécessaires pour pouvoir faire de la conversion des coordonnées géographiques à des coordonnées graphiques. Une fois que nous trouvons les coordonnées graphiques équivalentes d'un point d'intersection, nous utilisons la fonction « detectcorners » pour trouver la position graphique de l'intersection réelle dans l'image. L'interface de la fonction « detectcorners » est :

int detectcorners(PICINFO In, uint16 &Row, uint16 &Col)

En utilisant le point de référence [Row, Col] comme centre de la fenêtre d'analyse, cette fonction retourne le point d'intersection en terme de [Rangée, Colonne], placées dans la variable Row et Col également. Lorsque cette fonction ne peut détecter aucune intersection, elle retournera 0, auquel cas les variables Row, Col n'ont aucune signification.

5.3 RÉSULTAT DE DÉTECTION DES QUATRE COINS EXTRÊMES DE CARTE

Nous venons de voir les interfaces nécessaires pour détecter automatiquement les coins de rues et les coins extrêmes de carte. Dans cette section, nous allons montrer la performance de nos algorithmes, en général, pour la détection de ces coins. Pour cela, nous possédons au total 27 images de test avec lesquelles nous trouvons une variété des situations. Les procédures de test sont simples : nous n'avons qu'à ouvrir ces fichiers, un fichier à la fois, les emmagasiner dans un tampon de format PICINFO et appelle la fonction « detect4corners ». Pour ces tests, la fonction « detect4corners » va afficher à l'écran le résultat de détection comme dans l'exemple suivant :

```
Welcom to Corner Detector Program
        Done by Giang, Do-Tien
    Laboratoire Scribens ECOLE POLYTECHNIQUE
          CRIM et M3i

File name is :'sp1083n.tif'
The image width is 1280
The image height is 878
It has this message to you:"TIFF, 1-bit, min-is-black format.  (48507 bytes)"
Test for the color of the backgroud

Up left corner, Row = 42, Col = 20

Bottom right corner, Row = 854, Col = 1233

Bottom left corner, Row = 852, Col = 20

Up right corner, Row = 44, Col = 1232
```

Les résultats semblables pour les 27 images sont affichés dans le tableau 5.1 selon la convention suivante :

- Les données sont affichées selon trois grosseurs (grosse, moyenne et petite). Les gros caractères désignent que la donnée correspondant n'est pas exacte, car on a observé dans l'image qu'il y a trop de points qui répondent aux critères de

coins extrêmes. La figure 5.2 illustre ce fait en présentant une portion (le coin en haut à gauche) de la carte « sp0091.tif ».

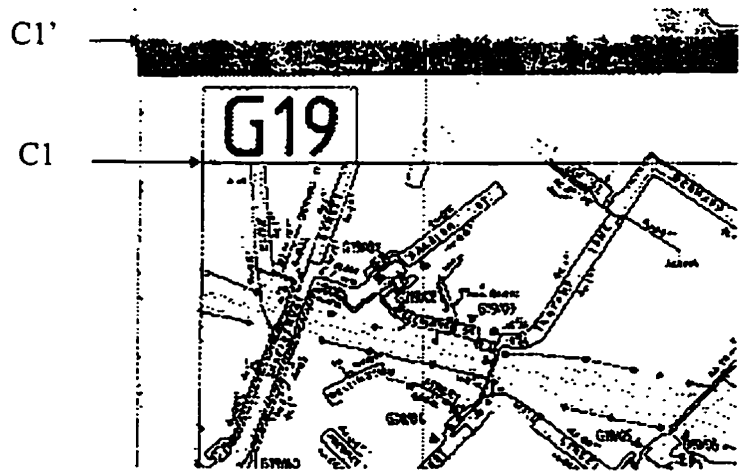


Figure 5.2 ***L'ambiguïté dans la détermination du coin extrême de carte***

Il est évident que, pour nous, le coin extrême de cette carte doit être située au point C1. Cependant, à cause du bruit, il est logique que notre fonction "detect4corners" détecte le point C1', car logiquement parlant, ce point est le coin extrême de l'image.

Les caractères à grosseur moyenne désignent que la donnée correspondante est exacte et qu'un bon détecteur de coin n'est pas permis de se tromper. Les petits caractères désignent que la donnée correspondant n'est pas exposée de façon évident. La figure 5.3 illustre ce phénomène en présentant une portion (coin en bas à gauche) de la carte « sp0693s.tif »

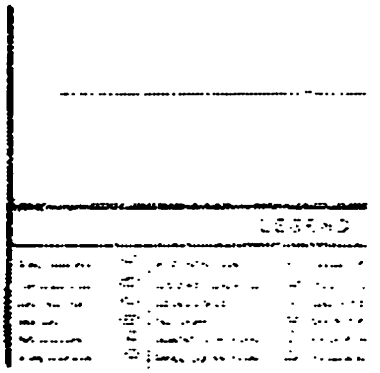


Figure 5.3 ***Ambiguïté dans la détermination du coin extrême de l'image***

Comme on peut voir, la bordure horizontale en bas de cette carte n'existe pas à cause du bruit. Dans ce cas, on ne peut pas déterminer la position du coin extrême de carte, même en se servant de notre système visuel. C'est pourquoi, nous avons décrit ce genre de situation par le terme « Ambigu et flou ».

Il y a des cartes qui ne possèdent pas tous les quatre coins comme on peut voir à la figure 5.4. Nous avons considéré le coin en haut à gauche et le coin en bas à gauche comme inexistants et les classifie dans la catégorie « N'existe pas » dans le tableau 5.1.

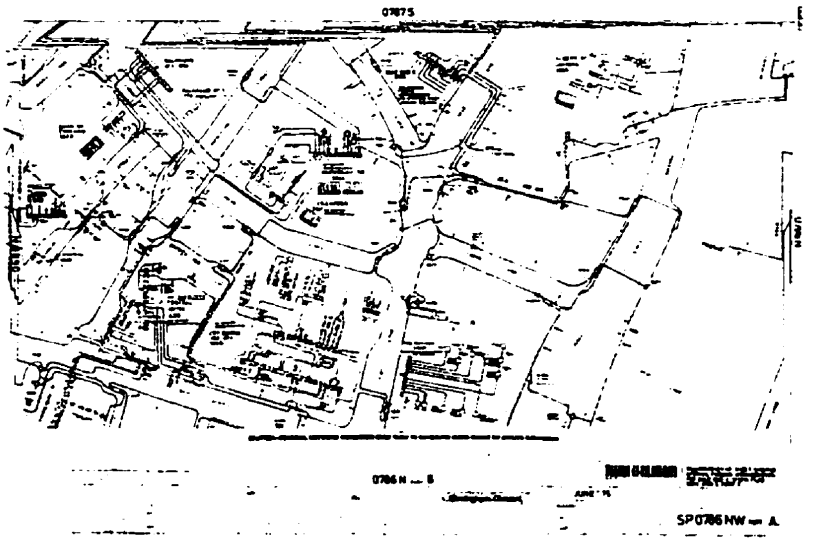


Figure 5.4 ***Illustration du cas où les coins C0 et C1 n'existent pas (ou ne sont pas bien définis)***

Il existe aussi des cas où le vrai coin de carte n'existe pas, mais plusieurs petits segments horizontaux et verticaux y sont présents de telle sorte que plusieurs petits coins non formels sont formés. Devant une telle situation, la fonction "detect4corners" détectera évidemment un coin qui, pourtant, ne devrait pas être considéré comme un coin extrême de la carte. Ce phénomène a été défini par le nom "Ambigu" dans le tableau de résultats.

Tableau 5.1 Résultat de détection des quatre coins extrêmes de la carte utilisant la fonction « detect4corners »

Nom du fichier	C1		C3		C0		C2		Err. Idéale	Err. totale
sp000912.tif	121	15	956	1185	961	13	110	1143		
	121	15	973	1190	964	12	164	1195		
Erreur	0	0	17	5	3	1	54	52	1,5811	23,962
sp004910.tif	113	11	958	1187	941	24	90	1104		
	113	11	958	1190	943	23	134	1187		
Erreur	0	0	0	3	2	1	44	83	1,7454	24,794
sp004912.tif	130	12	966	1193	970	17	98	1109		
	130	12	969	1193	Ambigu		145	1195		
Erreur	0	0	3	0	####	####	47	86	1,5	33,668
sp0091.tif	22	7	850	1225	853	34	7	1241		
	82	37	852	1224	849	33	45	1226		
Erreur	60	30	2	1	4	1	38	15	3,1796	28,574
sp0693s.tif	125	76	791	1200	855	76	126	1166		
	127	76	Ambigu et flou		856	76	127	1167		
Erreur	2	0	0	0	1	0	1	1	1,1036	1,1036
sp0487n.tif	137	71	790	1182	851	70	136	1163		
	137	71	Ambigu et flou		860	70	138	1162		
Erreur	0	0	0	0	9	0	2	1	0,7454	2,809
sp0693s.tif	42	13	842	1253	893	15	35	1252		
	43	13	867	1255	873	15	37	1251		
Erreur	1	0	25	2	20	0	2	1	1,618	12,079
sp0786n.tif	Distorsion		971	1006	968	61	307	1006		
	310	0	971	1007	969	0	308	1006		
Erreur	0	0	0	1	1	61	1	0	1	15,752
sp0786na.tif	167	44	953	1217	Distorsion		149	1216		
	Ambigu		954	1217	N'existe pas		139	1216		
Erreur	0	0	1	0	0	0	10	0	1	2,75
sp0786nb.tif	147	58	867	1181	866	12	66	1221		
	Ambigu et flou		869	1240	866	12	57	1234		
Erreur	0	0	2	59	0	0	9	13	0	18,711

Nom du fichier (cont.)	C1		C3		C0		C2		Err. Idéale	Err. totale
sp0892s.tif	26	40	847	1266	856	15	40	1270		
	25	40	865	1265	857	14	100	1236		
Erreur	0	0	18	1	0	0	0	0	0	4,5069
sp0895-s.tif	133	71	870	1163	Distorsion		133	1161		
	134	70	870	1163	859	65	134	1160		
Erreur	1	1	0	0	####	####	1	1	0,9428	0,9428
sp0990s.tif	18	27	867	1272	869	26	Distorsion			
	19	27	870	1273	870	26	21	1234		
Erreur	1	0	3	1	1	0	####	1234	1,7208	1,7208
sp0991n.tif	69	40	Distorsion		896	41	Distorsion			
	68	40	Ambigu		896	41	66	1279		
Erreur	1	0	####	0	0	0	####	1279	0,5	0,5
sp1081n.tif	31	17	847	1225	847	16	33	1226		
	33	17	847	1224	845	16	34	1225		
Erreur	2	0	0	1	2	0	1	1	1,6036	1,6036
sp1083n.tif	42	20	854	1233	852	20	44	1232		
	43	20	855	1234	856	20	42	1233		
Erreur	1	0	1	1	4	0	2	1	2,1626	2,1626
sp1083s.tif	Distorsion		867	1240	889	31	74	1241		
	74	32	848	1240	890	31	74	1242		
Erreur	####	32	19	0	1	0	0	1	1	7
sp1091nw.tif	146	75	858	1165	804	78	138	1159		
	147	75	858	1165	858	78	138	1158		
Erreur	1	0	0	0	54	0	0	1	0,6667	14
sp1195-s.tif	132	79	858	1206	850	75	135	1169		
	134	79	Ambigu		Ambigu		136	1169		
Erreur	2	0	0	0	####	75	1	0	1,5	1,5
sp1295-n.tif	137	75	861	1167	855	76	132	1163		
	137	75	863	1167	856	76	133	1163		
Erreur	0	0	2	0	1	0	1	0	1	1
sp1295-s.tif	131	79	855	1134	849	74	Distorsion			
	133	79	Ambigu		Ambigu		134	1170		
Erreur	2	0	0	0	####	####	####	####	2	2
sp1297-n.tif	59	72	788	1196	783	74	55	1164		
	60	73	Ambigu		784	75	57	1167		
Erreur	1	1	####	####	1	1	2	3	2,1447	2,1447
sp1297-s.tif	130	67	869	1163	Distorsion		131	1165		
	130	67	870	1163	859	65	132	1165		
Erreur	0	0	1	0	####	####	1	0	0,6667	0,6667
sp1298-n.tif	131	72	868	1172	Distorsion		132	1174		
	131	72	868	1173	858	69	132	1174		
Erreur	0	0	0	1	####	69	0	0	0,3333	0,3333

Nom du fichier (cont.)	C1		C3		C0		C2		Err. Idéale	Err. totale
sp1298-s.tif	61	76	776	1173	781	77	54	1168		
	62	76	778	1173	782	77	55	1168		
Erreur	1	0	2	0	1	0	1	0	1.25	1.25
sp2282.tif	28	22	858	1271	853	19	29	1272		
	28	23	859	1274	853	19	30	1272		
Erreur	0	1	1	3	0	0	1	0	1.2906	1.2906
sp244760.tif	185	18	996	1178	997	20	188	1177		
	185	18	997	1175	998	20	188	1177		
Erreur	0	0	1	3	1	0	0	0	1.0406	1.0406
								Erreur moyenne	1.2421	7.8891

Trois lignes sont associées à chaque image. La première ligne affiche les résultats de détection de la fonction "detect4corners". La deuxième ligne, en caractère gras, affiche les positions graphiques des coins observables sur l'image. À ce sujet, nous avons les cas "N'existe pas", "Ambigu et flou", "Ambigu", les gros caractères, les caractères moyens, et les petits caractères. La colonne "Err. Idéale" calcule l'erreur moyenne entre les résultats réalisés par la fonction "detect4corners" et les résultats présentés à la deuxième ligne pour les coins extrêmes qui sont facilement détectés à l'œil, et ce, sans ambiguïté d'interprétation. La colonne "Err. Totale" calcule l'erreur moyenne entre les deux séries de résultats pour tous les coins extrême qui peuvent être détectés à l'oeil. Évidemment, les cas "Ambigu", "Ambigu et flou" et le cas "N'existe pas" ne peuvent pas être tenus compte dans le calcul de l'erreur moyenne. La dernière ligne donne la moyenne de toutes les erreurs moyennes. Nous remarquons que si nous considérons uniquement les cas idéaux, cette valeur est de 1.2421 pixels. Pour tous les cas en général, l'erreur moyenne s'élève à 7.8891 pixels.

Dans la colonne de l'erreur idéale, les valeurs affichées en caractère gras signifient que la fonction "detect4corners" a atteint une performance remarquable, car elle a pu détecter correctement les positions des coins, même si ces coins sont difficilement visibles ou s'il y a trop de pixels d'événement qui entourent ces coins.

Comme on peut voir, la performance de la fonction "detect4corners" est assez remarquable. Cependant, cette fonction commet encore des erreurs inacceptables pour être utilisée de façon autonome (sans intervention des opérateurs en temps réel ou en temps différé). Nous discuterons, dans les prochaines sections, plus en détail la performance de cette fonction et celle de la fonction "detectcorners". Pour commencer, nous présentons les résultats de détection de la fonction "detectcorners".

5.3 RÉSULTAT DE DÉTECTION DES INTERSECTIONS DE RUES

Pour les images de test, nous utilisons deux fichiers (images) suivants: sp0091a.tif, sp0693s.tif. Pour simplifier la présentation, nous supposons que nous connaissons les positions graphiques approximatives de toutes les intersections de rues qui se trouvent dans l'image. La méthode dont on utilise pour attribuer des valeurs approximatives aux positions graphiques des intersections de rues consiste à ouvrir le fichier "*.tif" par un outil de dessin tel que CorelPHOTO-PAINT!©1993. Ensuite, on note la position graphique de ces intersections de rues en pointant le souris à ces endroits.

Avec l'image "sp0091a.tif", nous exécutons la fonction "detectcorners" utilisant les valeurs de position graphique des intersections de rues que nous avons mesuré avec le souris. Avec l'image "sp0693s.tif", nous ajoutons à chacun des points d'intersection, que nous avons mesuré avec le souris, un facteur de bruit aléatoire équivalent à ± 10 pixels dans les deux sens et nous exécutons la fonction "detectcorners" pour voir si cette fonction est capable de retrouver les points d'intersection originaux.

5.3.1 Résultats de détection des intersections de rues de l'image sp0091a.tif

Nous voyons dans la figure ci-bas l'image de la carte sp0091a.tif. Dans cette figure, nous présentons les valeurs des coordonnées graphiques [Rangée, Colonne] de certaines intersections de rues qui s'y trouvent.

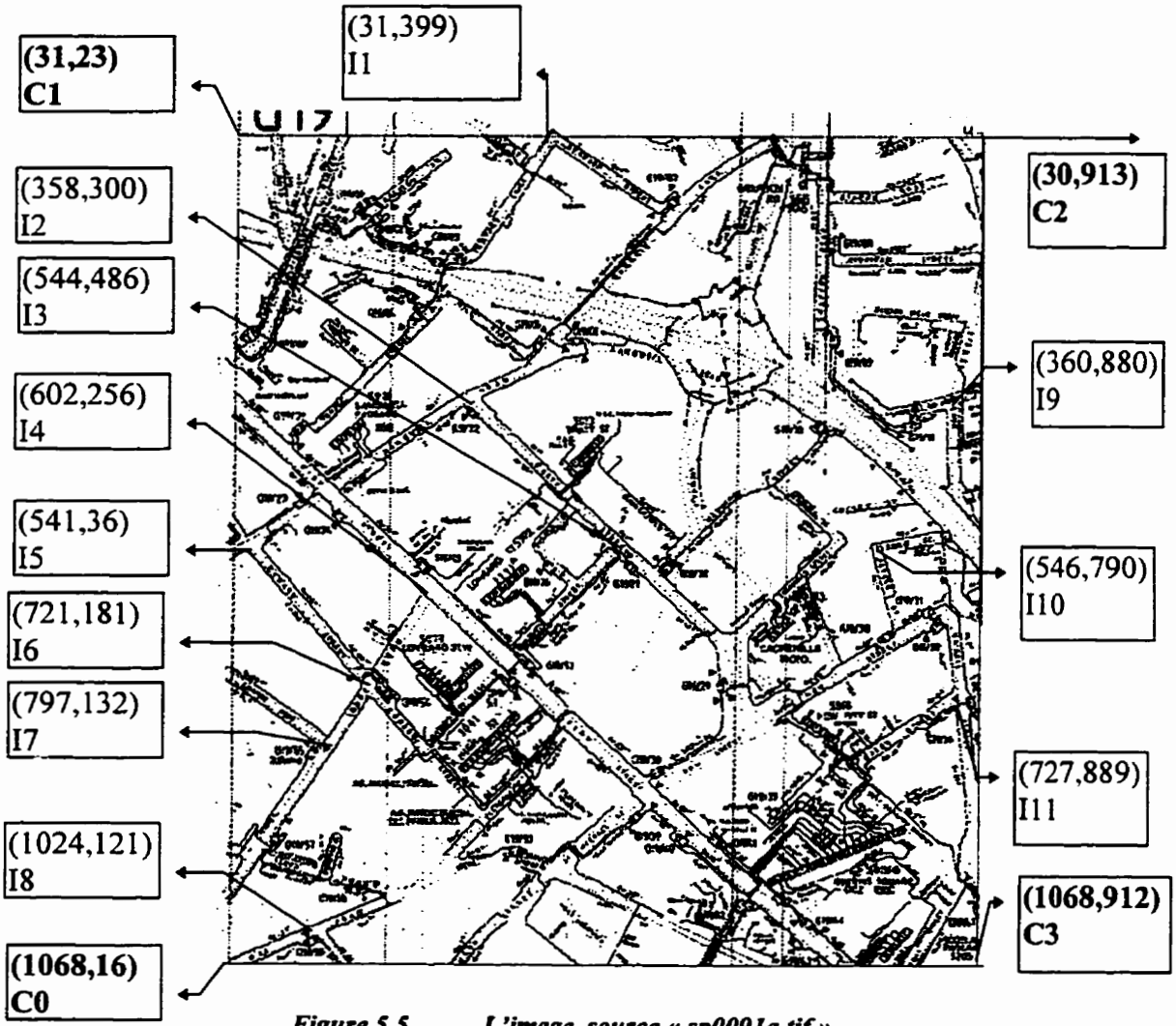


Figure 5.5 L'image source « sp0091a.tif »

Pour ce premier exemple, les valeurs des coordonnées graphiques mesurées avec le souris ne sont pas perturbées. Le tableau 5.2 montre les résultats de détection de la fonction "detectcorners".

Tableau 5.2 Résultats de détection des intersections de rues de la carte sp0091a.tif

	Rangée (observée)	Colonne (observée)	Bruit en y	Bruit en x	Rangée finale	Colonne finale	Rangée (détectée)	Colonne (détectée)	Erreur
I1	31	399	0	0	31	399	32	394	5.09902
I2	358	300	0	0	358	300			######
I3	544	486	0	0	544	486	543	483	3.16227
I4	602	256	0	0	602	256			#######
I5	541	36	0	0	541	36	542	32	4.12310
I6	721	181	0	0	721	181	706	185	15.5242
I7	797	132	0	0	797	132	798	133	1.41421
I8	1024	121	0	0	1024	121	1041	116	17.7200
I9	360	880	0	0	360	880			#######
I10	546	790	0	0	546	790	546	789	1
I11	727	889	0	0	727	889			#######
							Erreur moyenne		6.86326

Dans cet exemple, nous avons fixé le paramètre M, introduit à la section 4.11, à 3. Ainsi, nous identifions cette stratégie par le nom de règle de « trois fois ». Selon cette règle, lorsque la fonction « CornerDetect » détecte 3 fois le même point, ce point est alors considéré comme un point d'intersection de rues. L'intersection I4 n'a pas été détectée avec succès cas il y a trop de pixels bruités autour d'elle auquel cas il vaut mieux de dire non que de dire oui. Tandis que l'intersection I6 est plus bruitée par rapport aux autres intersections, ce qui explique la distance élevée entre les valeurs mesurées et les valeurs détectées. Il faut remarquer que la fonction "detectcorners" a plus de chance de faire des erreurs de détection pour des intersections qui se trouvent près des bordures de carte, car dans ces cas, la méthode de déplacement graduel de la fenêtre d'analyse ne fait pas déplacer, de façon efficace, cette fenêtre. En effet, à cause du phénomène de traitement des cas frontaliers, discutés à la section 4.11, la fenêtre d'analyse risque de ne pas pouvoir déplacer davantage. Ce qui donne que même après plusieurs tentatives de déplacement de la fenêtre d'analyse, les résultats de détections ne seront pas plus acceptables.

En effet, on aurait pu améliorer nos résultats de détection pour les cas de I6 et I8, si l'on applique plus correctement la règle de « trois fois ». Ce que nous avons fait actuellement, dans notre implantation, est que nous avons appelé la fonction « labelcoin » pour classifier chaque nouveau point d'intersection détecté. Lorsque cette fonction retourne 0, signifiant que le point détecté est déjà représenté par un point quelconque parmi des points emmagasinés précédemment dans le tableau de coins, on incrémente un compteur i. Après les cinq itérations, lorsque le compteur i a une valeur plus grande ou égale à la valeur de M (M fixé à 3 dans cet exemple), on conclut que le point détecté est un point d'intersection. Ce qui n'est pas totalement valide dans cette implantation est que le compteur i ne représente pas réellement le nombre de fois qu'un point particulier est détecté. En fait, ce compteur représente tout simplement le nombre de fois dont la fonction « labelcoin » réorganise le tableau de coin. Or, c'est ce qui est arrivé à la détection des points I6 et I8. En effet, le nombre de fois qu'on a détecté le point I6 et le point I8 est de 2 seulement, mais à cause que nous avons également détecté un autre point (près de I6 et près de I8) deux fois, ce qui fait que la valeur de compteur i de chaque cas dépasse 3.

On peut se demander si l'algorithme de détection des intersections de rues est encore fiable lorsqu'on fournit des points de départ de façon bruitée ou lorsque les rues à traiter n'ont pas toutes les propriétés souhaitées telles que vue à la section 4.9.3. C'est ce que nous allons voir dans les prochaines sections.

5.3.2 Résultats de détection des intersections de rues de l'image sp0693s.tif

Nous voyons dans la figure ci-bas l'image de la carte sp0693s.tif. Dans cette figure, nous présentons les valeurs des coordonnées graphiques [Rangée, Colonne] de certaines intersections de rues qui s'y trouvent.

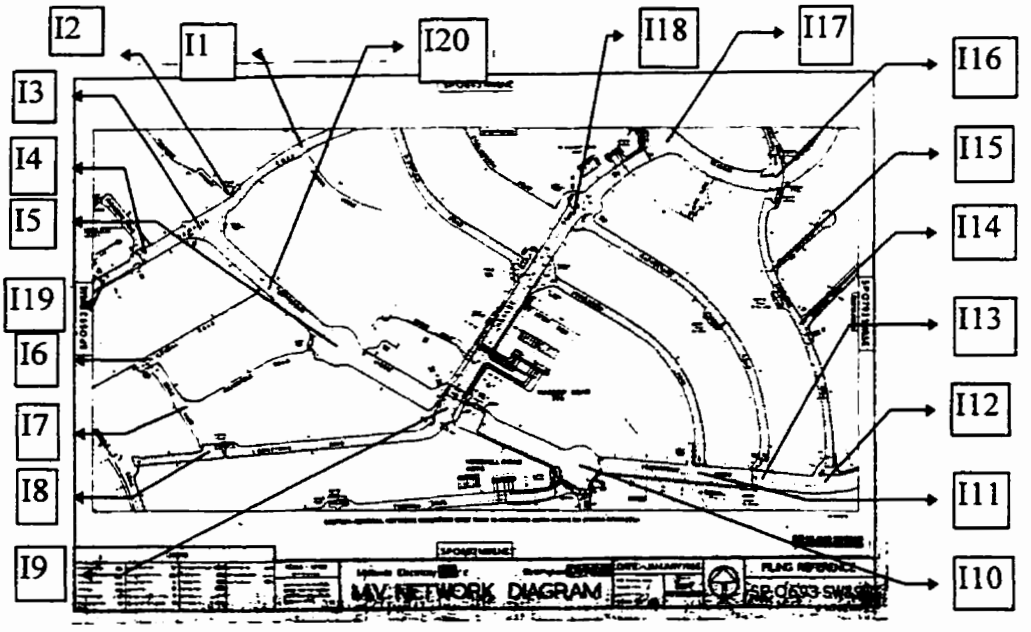


Figure 5.6 L'image source « sp0693s.tif »

Pour cet exemple, les valeurs des coordonnées graphiques mesurées avec le souris sont perturbées de façon aléatoires. Le tableau 5.3 montre les résultats de détection de la fonction "detectcorners".

Tableau 5.3 Résultats de détection des intersections de rues de la carte sp0693s.tif

	Rang obser	Colon obser	Bruit en y	Bruit en x	Rangée finale	Colonne finale	Rangdét. M=3	Colondét. M=3	Erreur	Rang dét. M=2	Colondét. M=2	Erreur
I1	154	366	1	1	155	367			######	153	363	3.162278
I2	237	258	3	-2	240	256			######	226	269	15.55635
I3	275	220	7	-5	282	215			######	283	190	31.04835
I4	316	128	-5	-1	311	127			######	316	130	2
I5	463	424	9	8	472	432			######			########
I6	482	131	2	-8	484	123			######	491	115	18.35756
I7	558	184	-7	3	551	187			######	554	169	15.52417
I8	631	225	6	4	637	229			######			########
I9	562	593	4	4	566	597			######			########
I10	651	791	-2	0	649	791			######			########
I11	660	975	-1	-3	659	972			######	659	966	9.055385
I12	678	1185	-6	0	672	1185			######	680	1192	7.28011
I13	671	1078	5	-10	676	1068			######			########
I14	432	1138	1	-8	433	1130			######	452	1146	21.54066
I15	346	1091	7	5	353	1096			######	372	1098	26.92582

(cont.)	Rang obser	Colon obser	Bruit en y	Bruit en x	Rangée finale	Colonne finale	Rangdét. M=3	Colondét. M=3	Erreur	Rang dét. M=2	Colondét. M=2	Erreur
I16	201	1100	-8	-6	193	1094			######	211	1111	14.86607
I17	139	928	6	0	145	928			######			#######
I18	258	781	5	-7	263	774			######	261	775	6.708204
I19	345	72	10	-1	355	71			######	351	69	6.708204
I20	369	319	7	-3	376	316	373	321	4.4721	373	321	4.472136
										Err. Moy.		13.08609

Dans cet exemple, nous illustrons le cas où M égal à 3 et M égal à 2 respectivement. On peut remarquer que le paramètre M représente le degré de sévérité de notre décision. Plus M est grand, plus on exige que la réponse doit être stable. Évidemment, pour M égal à 3, les réponses sont plus stables et plus exactes. Par conséquent, pour une image bruitée, le nombre de réponses obtenu va être moins élevé que dans le cas où on a M égal à 2. En revanche, avec M égal à 2, les réponses obtenues sont moins précises.

Dans cet exemple, nous avons aussi présenté les cas où les rues risquent de ne pas avoir des propriétés comme souhaitées telles que présentées à la section 4.9.3. Les intersections I5 et I10 dans la figure 5.6 constituent deux exemples. Ce qui explique le fait que notre détecteur n'est pas capable de détecter ces intersections même en réduisant le degré de sévérité.

5.4 DISCUSSION DE LA PERFORMANCE GÉNÉRALE DE NOS ALGORITHMES

Tout d'abord, il faut admettre que l'utilisation des tableaux pour gérer les données (éléments de Hough et les points d'intersections) diminuent énormément la vitesse d'exécution de notre programme. En effet, le temps moyen requis pour détecter un coin extrême de carte est mesuré à 3 secondes et le temps moyen requis pour détecter un point d'intersection de rues est égal à 28,65 secondes. Ces temps ne sont pas très considérables une fois que l'on remplace les tableaux par les listes dynamiquement liées.

un point d'intersection de rues est égal à 28,65 secondes. Ces temps ne sont pas très considérables une fois que l'on remplace les tableaux par les listes dynamiquement liées.

Avec notre méthode d'implantation actuelle, l'espace de mémoire requis est très raisonnable, malgré le fait que nous utilisons les tableaux, car en moyenne, nous utilisons jusqu'à 85 % de ces espaces préservés. Ce qui nous préoccupe le plus, c'est la précision de notre détecteur. On peut laisser les ordinateurs à exécuter tranquillement les procédures de reconnaissance des points d'intersections de rues mais on ne peut pas tolérer les machines de faire des erreurs de détection. Les erreurs de l'ordre de 6,863262 dans le cas de carte plus ou moins idéale et 13,08609 pour le cas de carte un peu moins idéale nous exige de penser à faire des améliorations. Or, d'après nos analyses, nous sommes certains qu'en appliquant correctement la règle de « trois fois » discutée plus haut, soit au niveau de la fonction «labelcoin » ou au niveau de la fonction « detectcorners », nous pouvons arriver à éliminer tous les candidats qui donnent lieu à des graves erreurs de détection. Faute de temps, nous devons arrêter notre travail à cette étape.

Au point de vue d'utilisation, malgré que nos algorithmes de détection des coins comprennent plusieurs modules (35), les concepteurs en logiciel peuvent facilement importer ces algorithmes dans leur programme en utilisant le fichier « improces.cpp », « iitdetect.cpp » et « improces.h ». Le fichier « improces.cpp » contient 1500 lignes de codes écrites en C++ et concerne les algorithmes de traitement et de détection des coins. Le fichier « iitdetect.cpp » contient les principales interfaces telles que la fonction « detect4corners » et la fonction « detectcorners ».

Dans ce chapitre, nous n'avons pas faire de l'analyse en tenant compte des coordonnées géographiques des intersections de rues, car nous voulons surtout tester la performance des nos algorithmes en matière de détection des intersections. De plus, nous n'avons pas encore fait des tests complets et élaborés avec l'équipe du CRIM, car nous

n'avons pas finalisé l'intégration au CRIM. Toutefois, l'équipe du CRIM et l'entreprise M3i voient en nos algorithmes de très forte capacité d'être utilisés dans diverses applications industrielles.

5.5 CONCLUSIONS

D'après ce qu'on peut observer, les résultats de détections permet de dire que nos algorithmes sont remarquables en matière de détection des lignes horizontales et verticales. Pour la détection des points d'intersection de rues, ils ne sont pas au point de distinguer les lignes principales des fausses lignes lorsque les fausses lignes sont également considérées comme les lignes principales, lorsqu'elles satisfont tout simplement aux contraintes dont on a fixé dans le chapitre précédent en ce qui concerne les propriétés intrinsèques d'une ligne principale.

Malgré que nos algorithmes sont testés avec l'image « sp0091a.tif » et « sp0693s.tif » seulement, nous ne perdons pas du sens de la généralité. Nous avons vu que nos algorithmes peuvent être très fiables pour être utilisés en industrie surtout si nous mettons plus de temps à réviser l'application de la règle de « trois fois ». Dans le pire cas, nous pouvons toujours fixer M égal à 3 pour qu'aucun mauvais coin soit admissible. Cependant, la sévérité exagérée peut nous conduire à des situations où nous n'obtenons pas assez de points de contrôle pour la prochaine étape qui est l'étape d'alignement de l'image.

En conclusion, on peut dire que même avec les meilleurs outils de traitements d'image, on doit vivre toujours avec le problème d'interprétation. On est même très conscient que notre façon d'interpréter des résultats n'est qu'une parmi plusieurs façons possibles.

Chapitre 6

Conclusions

Le traitement automatique des cartes géographiques exige d'abord la possibilité de convertir des coordonnées graphiques d'un point à des coordonnées géographiques équivalentes et vice versa. Il est insuffisant de se contenter simplement avec la reconnaissance d'un objet particulier sur la carte, en ignorant la position géographique de l'objet, car il est souvent question d'exprimer la position graphique de l'objet par rapport à d'autres cartes connexes. Pour cela, il faut s'assurer également que toutes les cartes contiennent moins de distorsion possible. Connaissant cette propriété importante, nous avons commencé notre projet par la détection des quatre coins extrêmes de carte et nous avons exigé que les valeurs des coordonnées géographiques de ces quatre coins soient disponibles. Ces coins constituent pour nous des points de référence que nous avons utilisé pour estimer les positions graphiques des intersections de rues sur la carte. Nous avons servi de ces points pour détecter des intersections de rues qui sont visibles à ces endroits.

Pour réaliser ce projet, nous avons appliqué le modèle d'énergie pour détecter les contours. D'après les paramètres que nous avons choisi, ce modèle peut être considéré aussi comme un outil d'amincissement. Nous avons également initié une nouvelle méthode d'implantation de la transformée de Hough qui réduit l'espace de mémoire requis sans perdre de la précision, car nous avons évité de faire de la quantification. Pour interpréter la réponse finale, nous avons servi de trois raisonnements philosophiques suivants :

1. Les points qui ont des caractéristiques communs doivent se réunir graduellement pour construire un entité représentatif et unique,

2. Lorsqu'il faut choisir un candidat unique parmi un ensemble de candidats, il faut commencer avec un critère (seuil) de sévérité élevé. Ce seuil doit être diminué à un rythme plus ou moins grand pour permettre aux candidats de deuxième ou troisième place d'avoir la chance de montrer leur qualité,

3. Un même phénomène qui se manifeste plus qu'un certain nombre de fois en dedans d'une petite intervalle nécessite d'être considéré plus particulièrement (pour être rejeté ou accepté).

6.1 APPORTS ET CONTRIBUTIONS ORIGINALES

Ce mémoire constitue un condensé important tant au niveau des lois mathématiques qu'au niveau physiologiques du modèle de la perception humaine. Nous savons depuis longtemps que notre système visuel utilise des filtres pour détecter les contours. Cependant, pour appliquer correctement ce modèle dans le monde de la simulation, il nous faut des fondements mathématiques plus solides. C'est seulement après avoir combiné ces deux aspects que nous arrivons à construire un filtre numérique le plus optimal.

Ce mémoire introduit une nouvelle façon plus efficace d'appliquer la théorie de la transformée de Hough. À l'aide de cette nouvelle méthode, nous évitons de faire de la quantification inutile. De plus, cette méthode nous exige à réviser l'algorithme de regroupement des éléments de Hough. Ainsi, nous avons trouvé que l'erreur de regroupement des éléments de Hough est plus prononcée lorsque le point (x,y) qui intervient dans le calcul des paramètres (d,θ) est relativement trop loin de l'origine (0,0).

Grâce à ces deux outils innovateurs, en plus de quelques stratégies intéressantes pour éliminer davantage les éléments de Hough indésirables, nous croyons que nos détecteurs de coins pourraient devenir un des outils fondamentaux de tous les travaux de traitement automatiques des cartes géographiques.

6.2 PERSPECTIVES

Faute de temps, nous avons ouvert des chemins sans pouvoir y s'engager. La première chose à être améliorée dans notre programme est d'utiliser les structures de liste dynamiquement liée pour gérer plus efficacement des informations. Ce changement peut améliorer de beaucoup la vitesse d'exécution de notre programme, car nous utilisons régulièrement les fonctions de recherche « Exist », d'insertion « Insert » et de suppression « Delete ». Pour l'instant, le coût de recherche d'un élément ou le coût de suppression d'un élément, tel que implanté avec des tableaux, sont de O(n). De plus, la fonction « reorganize » et la fonction « reduceHough » qui font des appels excessifs aux fonctions « Exist » et « Delete », sont utilisées au moins un certain nombre de fois dans l'exécution du programme. Avec la structure de liste liée, le coût de recherche d'un élément reste encore à O(n) mais le coût de suppression d'un élément n'est que de O(1). En plus, avec une telle implantation, nous n'avons plus besoin de réorganiser le tableau de Hough, car il n'y a jamais de taille limite pour une liste liée dynamiquement.

Il y a aussi un petit problème avec l'algorithme « LocateCorner » qui consiste à déplacer la fenêtre d'analyse tant qu'on rencontre une zone contenant assez de pixels d'événement où on assume qu'il s'agit là d'une bordure extrême de la carte. Or, ceci n'est plus vrai dans le cas où il y a une discontinuité importante dans la bordure extrême de la carte de telle sorte que la fenêtre d'analyse peut dépasser cette bordure. Dans ce cas, la première zone, qui contient assez de pixels d'événement, que la fenêtre d'analyse détecte ne correspond à aucune bordure extrême.

Le troisième aspect à être amélioré est que, pour l'instant, l'algorithme de détection de contours utilise les filtres de 1D pour détecter un signal de 2D (image). Ceci a comme effet d'ajouter des pixels redondants ou d'enlever des pixels importants à cause des phénomènes de décalage de 1 pixel tels que discutés à la section 2.6.1.

Le denier défaut à discuter est au niveau de la détection des intersections pour les rues ayant les largeurs plus petites que 7 pixels ou ayant des propriétés qui ne satisfont pas aux contraintes imposées. Dans un tel cas, notre détecteur peut faire de graves erreurs (tableau 5.3, intersection I14, I15, etc) sans la règle de « trois fois ». Pour améliorer la chance de ne détecter que des intersections valides, nous croyons qu'il faut implanter plus correctement la règle de « trois fois », tout en réduisant le degré de sévérité M à 2.

Enfin, nous aimerons que les défauts cités ci-dessus seront résolus dans un jour très proche par ceux et celles qui nous suivent, pour que nous puissions faire des grands pas ensemble vers l'automatisation totale des processus de traitement et d'analyse d'images.

Bibliographie

AGHAZAN et HAMID K. (1994). SLIDE: subspace-base line detection, IEEE Transactions on Pattern Analysis and Machine Intelligence, 16(11).

ARDESHIR GOSHTASBY, (1987). Piecewise cubic mapping functions for image registration", Pattern Recognition 20, no 5, p. 525-533.

BURR DAVID C., M. CONCETTA MORRONE et DONATELLA SPINELLI, (1989). Evidence for edge and bar detectors in humain vision, Vision Res. 29, 419-431.

CANNY J., (1986). A computational Approach to Edge Detection, IEEE Transactions on Pattern Analysis and Machine Intelligence, 8(6), 679-698.

CHEE-WOO KANG et all., (1991). Extraction of straight line segments using rotation transformation: generalized Hough Transformation, Pattern Recognition, 24(7), 663-641.

CHEN C.-H., LEE J.-S., et SUN Y.-N. (1995). Wavelet transformation for grey-level corner detection, Pattern Recognition, 28(6), 853-861.

CHRISTIAN RONSE, (1993). On idempotence and related requirements in Edge Detection, IEEE Transactions on Pattern Analysis and Machine Intelligence, 15(5).

CRAIG A. LINDLEY, (1990). Practical image processing in C : acquisition, manipulation, storage.

DETLEF RUPRECHT et HEINRICH M., (1995). Image warping with scattered data interpolation, IEEE Computer Graphics and applications, 37-43.

DAVIES E.R., (1990). Machine Vision : theory, algorithms, practicalities, Academic press limited.

FERMULLER C. et KROPATSCH W. , (1994). A syntactic approach to scale-space-based corner description, IEEE Transactions on Pattern Analysis and Machine Intelligence, 16(7), 748-751.

HOUGH, P.V.C., (1962). Method and means for recognizing complex patterns, US Patent 3069654.

HUBEL D.H. et WIESEL T.N., (1977). Architecture of macaque monkey visual cortex, Proc. R. Soc. Lond. B198, 1-59.

LEE J.-S., SUN Y.-N., CHEN C.-H., et TSAI C.-T. (1993). Wavelet based corner detection, Pattern Recognition, 26(6), 853-865.

MARR D., (1982). Vision, Freeman Ed.

MARR D. et E.HILDRETH, (1980). Theory of edge detection, Proc. R. Soc. London B, 207, 187-217.

MEER P., BAUGHER E.S., et ROSENFELD A. (1988). Extraction of trend lines and extrema from multiscale curves. Pattern Recognition, 21(3), 217-226,.

MING ZHANG, (1996). On the discretization of parameter domain in Hough Transformation, Proceedings of ICPR, 527 - 531.

MORRONE M.C. & BURR D.C., (1989). Feature detection in human vision : a phase dependent energy model, Proc. R. Soc. Lond. B235, 221-245.

NACHMIAS J. et SANSBURY R. V., (1974). Grating contrast : discrimination may be better than detection, Vision Res. 14, 1039-1042.

NIBLACK W. et D.PETKOVIE, (1990). On improving the accuracy of the Hough Transform, Machine Vision and Applications, 3, 87-106.

OWENS, S. VENKATESH, et J. ROSS, (1989). Edge detection is a projection, Patt. Recogn. Lett., 9, 233-244.

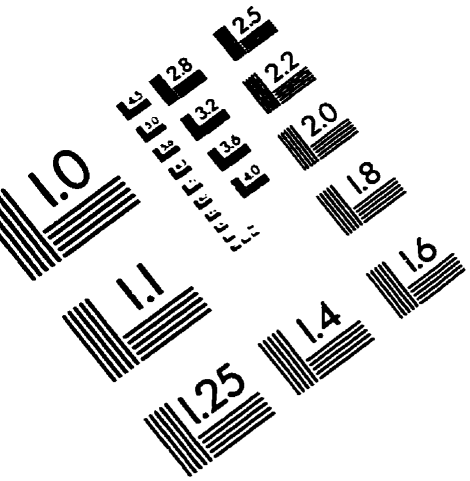
PARK, D.J.; NAM, K.M.; RAE-HONG PARK, (1995). Multiresolution edge detection techniques, Pattern Recognition, 28(2), 211-29.

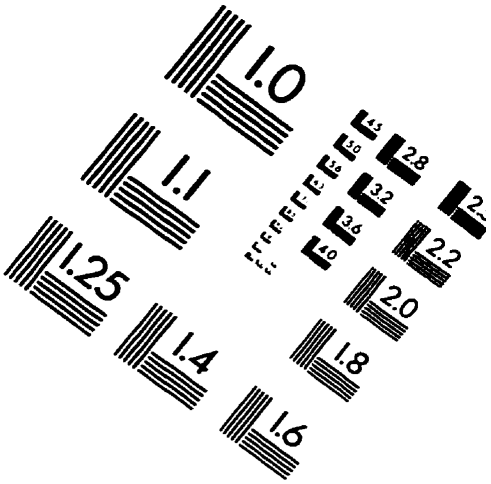
ROSENFELD A. et LINDA SHAPIRO, (1993). Computer vision and image processing, Academic Press, inc.

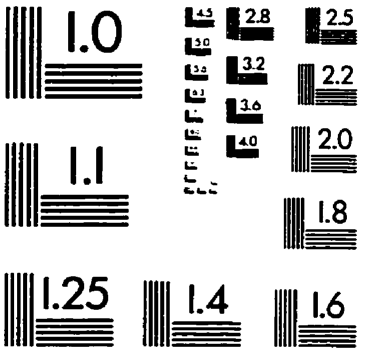
SHAPLEY R.M. et TOLHURST D.J., (1973). Edge detectors in humain vision, J. Physiol., Lond. 229, 165-183,.

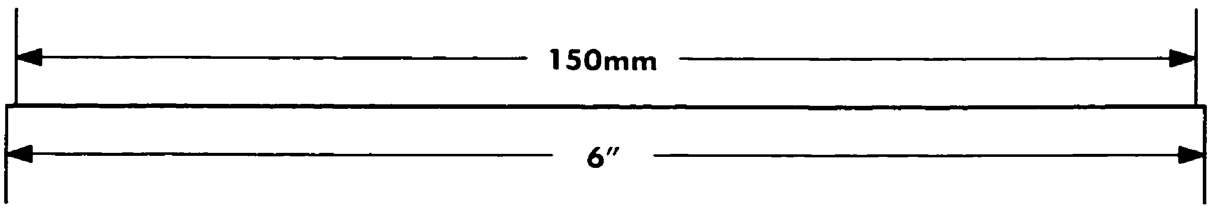
TOLHURST D.J., (1972). On the possible existence of edges detectors neurones in the humain visual cortex, Vision Res. 12, 797-804.

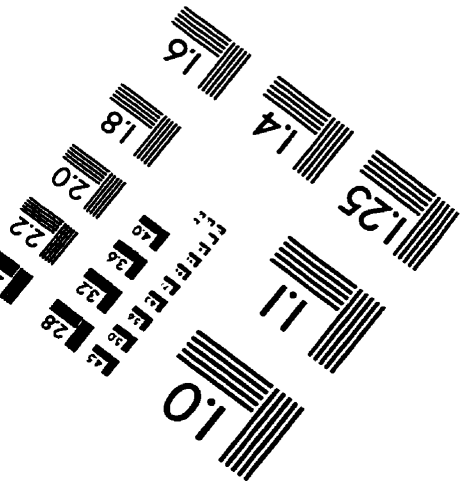
IMAGE EVALUATION TEST TARGET (QA-3)

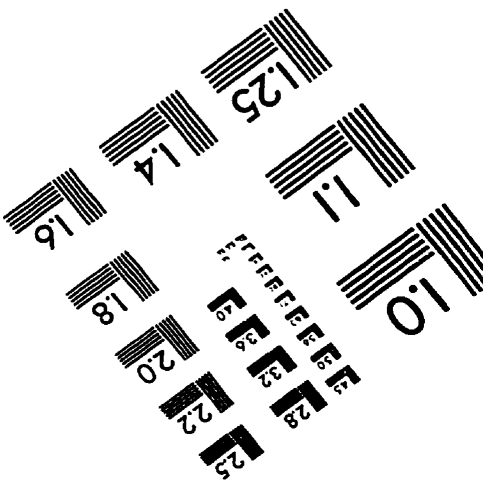












APPLIED IMAGE, Inc
1653 East Main Street
Rochester, NY 14609 USA
Phone: 716/482-0300
Fax: 716/288-5989